

TIBCO SmartSockets™

Application Programming Interface

*Software Release 6.8
July 2006*



Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE *TIBCO SMARTSOCKETS INSTALLATION GUIDE*). USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIB, TIBCO, Information Bus, The Power of Now, TIBCO Adapter, RTclient, RTserver, RTworks, SmartSockets, and Talarian are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and /or other countries.

EJB, J2EE, JMS and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the readme.txt file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND /OR CHANGES IN THE PRODUCT(S) AND /OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

Copyright © 1991–2006 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Contents

Preface	xxvii
Related Documentation	xxviii
TIBCO Product Documentation	xxviii
Using the Online Documentation	xxviii
Conventions Used in This Manual	xxix
Typeface Conventions	xxix
Notational Conventions	xxx
Identifiers	xxx
Case	xxx
How to Contact TIBCO Support	xxxii
 Chapter 1 Introduction	 1
Overview	2
Naming Conventions	3
Essential API Functions	4
Message Type Functions	4
Message Functions	4
Communication Functions	5
Utility Functions	5
Data Structures	6
Include Files	7
Compiling and Linking	8
On UNIX	8
On OpenVMS	9
On Windows	10
Messages	10
Message Types	12
Message Files	13
Connections	13
Connection Callbacks	14
Global Connections	17
Global Connection Callbacks	18

Multiple Connections	20
Multiple Connection Callbacks	20
Events	21
Properties Table	21
Error Handling	22
Chapter 2 TipcBuf*	23
TipcBufMsgAppend	24
TipcBufMsgNext	25
Chapter 3 TipcCbConnProcessGmdFailure - TipcConnWriteCbLookup	27
TipcCbConnProcessGmdFailure	30
TipcCbConnProcessKeepAliveCall	33
TipcCbSrvError	35
TipcCbSrvProcessControl	37
TipcCbSrvProcessGmdFailure	39
TipcConnAccept	41
TipcConnBufferGetReadSize	42
TipcConnBufferGetWriteSize	43
TipcConnCheck	44
TipcConnCreate	47
TipcConnCreateClient	49
TipcConnCreateServer	51
TipcConnDefaultCbCreate	53
TipcConnDefaultCbLookup	55
TipcConnDestroy	56
TipcConnErrorCbCreate	58
TipcConnErrorCbLookup	60
TipcConnFlush	61
TipcConnGetArch	63
TipcConnGetAutoFlushSize	65
TipcConnGetBlockMode	66
TipcConnGetGmdMaxSize	68
TipcConnGetGmdNumPending	70
TipcConnGetNode	71
TipcConnGetNumQueued	72
TipcConnGetPid	73

TipcConnGetSocket	74
TipcConnGetTimeout	75
TipcConnGetUniqueSubject	77
TipcConnGetUser	78
TipcConnGetXtSource	79
TipcConnGmdFileCreate	81
TipcConnGmdFileDelete	85
TipcConnGmdMsgDelete	87
TipcConnGmdMsgResend	90
TipcConnGmdResend	93
TipcConnKeepAlive	95
TipcConnLock	97
TipcConnMainLoop	98
TipcConnMsgInsert	100
TipcConnMsgNext	102
TipcConnMsgProcess	104
TipcConnMsgSearch	106
TipcConnMsgSearchType	110
TipcConnMsgSend	111
TipcConnMsgSendRpc	114
TipcConnMsgWrite	118
TipcConnMsgWriteVa	120
TipcConnProcessCbCreate	122
TipcConnProcessCbLookup	124
TipcConnQueueCbCreate	126
TipcConnQueueCbLookup	128
TipcConnRead	130
TipcConnReadCbCreate	133
TipcConnReadCbLookup	135
TipcConnSetAutoFlushSize	137
TipcConnSetBlockMode	138
TipcConnSetGmdMaxSize	139
TipcConnSetSocket	141
TipcConnSetTimeout	142
TipcConnTrafficGetBytesRecv8	144
TipcConnTrafficGetBytesSent8	145

TipcConnTrafficGetMsgsRecv8	146
TipcConnTrafficGetMsgsSent8	147
TipcConnUnlock	148
TipcConnWriteCbCreate	149
TipcConnWriteCbLookup	151
Chapter 4 TipcDeliveryModeToStr - TipcLbModeToStr	153
TipcDeliveryModeToStr	155
TipcDispatcherCreate	156
TipcDispatcherCreateDetached	157
TipcDispatcherDestroy	158
TipcDispatcherDispatch	159
TipcDispatcherMainLoop	160
TipcDispatcherSrvAdd	161
TipcDispatcherSrvRemove	162
TipcEventCreate	163
TipcEventCreateConn	165
TipcEventCreateMsg	167
TipcEventCreateMsgType	169
TipcEventCreateSocket	171
TipcEventCreateTimer	173
TipcEventDestroy	175
TipcEventGetCheckMode	176
TipcEventGetConn	177
TipcEventGetData	178
TipcEventGetDispatcher	179
TipcEventGetInterval	180
TipcEventGetSocket	181
TipcEventGetType	182
TipcEventSetInterval	183
TipcFtToStr	184
TipcGetGmdDir	186
TipcInitCommands	188
TipcInitThreads	189
TipcLbModeToStr	190

Chapter 5 TipcMon*	191
TipcMonClientBufferGetWatch	194
TipcMonClientBufferPoll	196
TipcMonClientBufferSetWatch	199
TipcMonClientCbPoll	203
TipcMonClientCongestionGetWatch	206
TipcMonClientCongestionSetWatch	207
TipcMonClientCpuPoll	210
TipcMonClientExtPoll	213
TipcMonClientGeneralPoll	216
TipcMonClientInfoPoll	221
TipcMonClientMsgRecvGetWatch	225
TipcMonClientMsgRecvSetWatch	227
TipcMonClientMsgSendGetWatch	231
TipcMonClientMsgSendSetWatch	233
TipcMonClientMsgTrafficPoll	236
TipcMonClientMsgTypeExPoll	240
TipcMonClientMsgTypePoll	245
TipcMonClientNamesGetWatch	250
TipcMonClientNamesNumPoll	252
TipcMonClientNamesPoll	254
TipcMonClientNamesSetWatch	256
TipcMonClientOptionPoll	260
TipcMonClientSubjectExPoll	264
TipcMonClientSubjectPoll	269
TipcMonClientSubscribeGetWatch	273
TipcMonClientSubscribeNumPoll	275
TipcMonClientSubscribePoll	278
TipcMonClientSubscribeSetWatch	281
TipcMonClientTimeGetWatch	284
TipcMonClientTimePoll	286
TipcMonClientTimeSetWatch	289
TipcMonClientVersionPoll	292
TipcMonExtDelete	294
TipcMonExtSetBinary	295
TipcMonExtSetBool	296

TipcMonExtSetBoolArray	297
TipcMonExtSetInt2	298
TipcMonExtSetInt2Array	299
TipcMonExtSetInt4	300
TipcMonExtSetInt4Array	301
TipcMonExtSetInt8	302
TipcMonExtSetInt8Array	303
TipcMonExtSetReal4	304
TipcMonExtSetReal4Array	305
TipcMonExtSetReal8	306
TipcMonExtSetReal8Array	307
TipcMonExtSetReal16	308
TipcMonExtSetReal16Array	309
TipcMonExtSetStr	310
TipcMonExtSetStrArray	311
TipcMonExtSetUtf8	312
TipcMonExtSetUtf8Array	313
TipcMonGetIdentStr	314
TipcMonPrintWatch	316
TipcMonProjectNamesGetWatch	317
TipcMonProjectNamesPoll	319
TipcMonProjectNamesSetWatch	321
TipcMonServerBufferPoll	324
TipcMonServerCongestionGetWatch	328
TipcMonServerCongestionSetWatch	330
TipcMonServerConnGetWatch	334
TipcMonServerConnPoll	336
TipcMonServerConnSetWatch	338
TipcMonServerCpuPoll	341
TipcMonServerGeneralPoll	344
TipcMonServerMaxClientLicensesGetWatch	349
TipcMonServerMaxClientLicensesSetWatch	351
TipcMonServerMsgTrafficExPoll	354
TipcMonServerMsgTrafficPoll	358
TipcMonServerNamesGetWatch	362
TipcMonServerNamesPoll	364

TipcMonServerNamesSetWatch	366
TipcMonServerOptionPoll	369
TipcMonServerRoutePoll	373
TipcMonServerStartTimePoll	376
TipcMonServerTimePoll	379
TipcMonServerVersionPoll	382
TipcMonSetIdentStr	384
TipcMonSubjectNamesGetWatch	385
TipcMonSubjectNamesPoll	387
TipcMonSubjectNamesSetWatch	389
TipcMonSubjectSubscribeGetWatch	392
TipcMonSubjectSubscribePoll	394
TipcMonSubjectSubscribeSetWatch	397
Chapter 6 TipcMsgAck - TipcMsgAddNamedXmlPtr	401
TipcMsgAck	403
TipcMsgAddNamedBinary	405
TipcMsgAddNamedBinaryPtr	406
TipcMsgAddNamedBool	408
TipcMsgAddNamedBoolArray	409
TipcMsgAddNamedBoolArrayPtr	410
TipcMsgAddNamedByte	412
TipcMsgAddNamedChar	413
TipcMsgAddNamedInt2	414
TipcMsgAddNamedInt2Array	415
TipcMsgAddNamedInt2ArrayPtr	417
TipcMsgAddNamedInt4	419
TipcMsgAddNamedInt4Array	420
TipcMsgAddNamedInt4ArrayPtr	422
TipcMsgAddNamedInt8	424
TipcMsgAddNamedInt8Array	425
TipcMsgAddNamedInt8ArrayPtr	427
TipcMsgAddNamedMsg	429
TipcMsgAddNamedMsgArray	430
TipcMsgAddNamedMsgArrayPtr	432
TipcMsgAddNamedMsgPtr	434

TipcMsgAddNamedReal4	435
TipcMsgAddNamedReal4Array	436
TipcMsgAddNamedReal4ArrayPtr	438
TipcMsgAddNamedReal8	440
TipcMsgAddNamedReal8Array	441
TipcMsgAddNamedReal8ArrayPtr	443
TipcMsgAddNamedReal16	445
TipcMsgAddNamedReal16Array	446
TipcMsgAddNamedReal16ArrayPtr	448
TipcMsgAddNamedStr	450
TipcMsgAddNamedStrArray	451
TipcMsgAddNamedStrArrayPtr	453
TipcMsgAddNamedStrPtr	455
TipcMsgAddNamedTimestamp	456
TipcMsgAddNamedTimestampArray	457
TipcMsgAddNamedTimestampArrayPtr	459
TipcMsgAddNamedUnknown	461
TipcMsgAddNamedUtf8	462
TipcMsgAddNamedUtf8Array	463
TipcMsgAddNamedUtf8ArrayPtr	465
TipcMsgAddNamedUtf8Ptr	467
TipcMsgAddNamedXml	468
TipcMsgAddNamedXmlPtr	469
Chapter 7 TipcMsgAppend*	471
TipcMsgAppendBinary	473
TipcMsgAppendBinaryPtr	475
TipcMsgAppendBool	477
TipcMsgAppendBoolArray	479
TipcMsgAppendBoolArrayPtr	481
TipcMsgAppendByte	483
TipcMsgAppendChar	485
TipcMsgAppendInt2	487
TipcMsgAppendInt2Array	489
TipcMsgAppendInt2ArrayPtr	491
TipcMsgAppendInt4	493

TipcMsgAppendInt4Array	495
TipcMsgAppendInt4ArrayPtr	497
TipcMsgAppendInt8	499
TipcMsgAppendInt8Array	501
TipcMsgAppendInt8ArrayPtr	503
TipcMsgAppendMsg	505
TipcMsgAppendMsgArray	507
TipcMsgAppendMsgArrayPtr	510
TipcMsgAppendMsgPtr	513
TipcMsgAppendReal4	515
TipcMsgAppendReal4Array	517
TipcMsgAppendReal4ArrayPtr	519
TipcMsgAppendReal8	521
TipcMsgAppendReal8Array	523
TipcMsgAppendReal8ArrayPtr	525
TipcMsgAppendReal16	527
TipcMsgAppendReal16Array	529
TipcMsgAppendReal16ArrayPtr	531
TipcMsgAppendStr	533
TipcMsgAppendStrArray	535
TipcMsgAppendStrArrayPtr	537
TipcMsgAppendStrPtr	539
TipcMsgAppendStrReal8	541
TipcMsgAppendTimestamp	543
TipcMsgAppendTimestampArray	545
TipcMsgAppendTimestampArrayPtr	547
TipcMsgAppendUnknown	549
TipcMsgAppendUtf8	551
TipcMsgAppendUtf8Array	553
TipcMsgAppendUtf8ArrayPtr	555
TipcMsgAppendUtf8Ptr	557
TipcMsgAppendXml	559
TipcMsgAppendXmlPtr	561

Chapter 8 TipcMsgClone - TipcMsgIncrRefCount.	563
TipcMsgClone	566
TipcMsgCreate	568
TipcMsgDeleteCurrent	570
TipcMsgDeleteField	571
TipcMsgDeleteNamedField	572
TipcMsgDestroy	573
TipcMsgExistsNamed	575
TipcMsgFieldSetSize	576
TipcMsgFieldUpdateBinaryPtr	577
TipcMsgFieldUpdateBoolArrayPtr	578
TipcMsgFieldUpdateInt2ArrayPtr	579
TipcMsgFieldUpdateInt4ArrayPtr	580
TipcMsgFieldUpdateInt8ArrayPtr	581
TipcMsgFieldUpdateMsgArrayPtr	582
TipcMsgFieldUpdateMsgPtr	583
TipcMsgFieldUpdateReal4ArrayPtr	584
TipcMsgFieldUpdateReal8ArrayPtr	585
TipcMsgFieldUpdateReal16ArrayPtr	586
TipcMsgFieldUpdateStrArrayPtr	588
TipcMsgFieldUpdateStrPtr	589
TipcMsgFieldUpdateTimestampArrayPtr	590
TipcMsgFieldUpdateUtf8ArrayPtr	591
TipcMsgFieldUpdateUtf8Ptr	592
TipcMsgFieldUpdateXmlPtr	593
TipcMsgFileCreate	594
TipcMsgFileCreateFromFile	596
TipcMsgFileDestroy	598
TipcMsgFileRead	599
TipcMsgFileWrite	601
TipcMsgGenerateMessageId	604
TipcMsgGetArrivalTimestamp	605
TipcMsgGetCompression	606
TipcMsgGetCorrelationId	607
TipcMsgGetCurrent	608
TipcMsgGetCurrentFieldKnown	610

TipcMsgGetDeliveryMode	612
TipcMsgGetDeliveryTimeout	614
TipcMsgGetDest	615
TipcMsgGetExpiration	616
TipcMsgGetHeaderStrEncode	617
TipcMsgGetLbMode	618
TipcMsgGetMessageId	620
TipcMsgGetNameCurrent	621
TipcMsgGetNamedBinary	622
TipcMsgGetNamedBool	624
TipcMsgGetNamedBoolArray	625
TipcMsgGetNamedByte	627
TipcMsgGetNamedChar	628
TipcMsgGetNamedInt2	629
TipcMsgGetNamedInt2Array	630
TipcMsgGetNamedInt4	632
TipcMsgGetNamedInt4Array	633
TipcMsgGetNamedInt8	635
TipcMsgGetNamedInt8Array	637
TipcMsgGetNamedMsg	639
TipcMsgGetNamedMsgArray	640
TipcMsgGetNamedReal4	642
TipcMsgGetNamedReal4Array	643
TipcMsgGetNamedReal8	645
TipcMsgGetNamedReal8Array	646
TipcMsgGetNamedReal16	648
TipcMsgGetNamedReal16Array	650
TipcMsgGetNamedStr	652
TipcMsgGetNamedStrArray	653
TipcMsgGetNamedTimestamp	655
TipcMsgGetNamedTimestampArray	657
TipcMsgGetNamedUnknown	659
TipcMsgGetNamedUtf8	660
TipcMsgGetNamedUtf8Array	661
TipcMsgGetNamedXml	663
TipcMsgGetNumFields	664

TipcMsgGetPacketSize 665

TipcMsgGetPriority 666

TipcMsgGetReadOnly 667

TipcMsgGetRefCount 668

TipcMsgGetReplyTo 669

TipcMsgGetSender 670

TipcMsgGetSenderTimestamp 671

TipcMsgGetSeqNum 672

TipcMsgGetType 673

TipcMsgGetTypeNamed 675

TipcMsgGetTypeNum 676

TipcMsgGetUserProp 677

TipcMsgIncrRefCount 679

Chapter 9 TipcMsgNextBinary - TipcMsgTraverse 681

TipcMsgNextBinary 683

TipcMsgNextBool 685

TipcMsgNextBoolArray 687

TipcMsgNextByte 689

TipcMsgNextChar 691

TipcMsgNextInt2 693

TipcMsgNextInt2Array 695

TipcMsgNextInt4 697

TipcMsgNextInt4Array 699

TipcMsgNextInt8 701

TipcMsgNextInt8Array 703

TipcMsgNextMsg 705

TipcMsgNextMsgArray 707

TipcMsgNextReal4 709

TipcMsgNextReal4Array 711

TipcMsgNextReal8 713

TipcMsgNextReal8Array 715

TipcMsgNextReal16 717

TipcMsgNextReal16Array 719

TipcMsgNextStr 721

TipcMsgNextStrArray 723

TipcMsgNextStrReal8	725
TipcMsgNextTimestamp	727
TipcMsgNextTimestampArray	729
TipcMsgNextType	731
TipcMsgNextUnknown	733
TipcMsgNextUtf8	735
TipcMsgNextUtf8Array	737
TipcMsgNextXml	739
TipcMsgPrint	741
TipcMsgPrintError	743
TipcMsgRead	745
TipcMsgReadVa	747
TipcMsgSetArrivalTimestamp	749
TipcMsgSetCompression	750
TipcMsgSetCorrelationId	752
TipcMsgSetCurrent	753
TipcMsgSetDeliveryMode	755
TipcMsgSetDeliveryTimeout	757
TipcMsgSetDest	758
TipcMsgSetExpiration	759
TipcMsgSetHeaderStrEncode	760
TipcMsgSetLbMode	761
TipcMsgSetNameCurrent	763
TipcMsgSetNumFields	764
TipcMsgSetPriority	766
TipcMsgSetReplyTo	767
TipcMsgSetSender	768
TipcMsgSetSenderTimestamp	769
TipcMsgSetType	770
TipcMsgSetUserProp	771
TipcMsgTraverse	772

Chapter 10 TpcMsgUpdateNamedBinary - TpcMsgWriteVa 775

TpcMsgUpdateNamedBinary 777

TpcMsgUpdateNamedBinaryPtr 778

TpcMsgUpdateNamedBool 780

TpcMsgUpdateNamedBoolArray 781

TpcMsgUpdateNamedBoolArrayPtr 782

TpcMsgUpdateNamedByte 784

TpcMsgUpdateNamedChar 785

TpcMsgUpdateNamedInt2 786

TpcMsgUpdateNamedInt2Array 787

TpcMsgUpdateNamedInt2ArrayPtr 788

TpcMsgUpdateNamedInt4 790

TpcMsgUpdateNamedInt4Array 791

TpcMsgUpdateNamedInt4ArrayPtr 792

TpcMsgUpdateNamedInt8 794

TpcMsgUpdateNamedInt8Array 795

TpcMsgUpdateNamedInt8ArrayPtr 796

TpcMsgUpdateNamedMsg 798

TpcMsgUpdateNamedMsgArray 799

TpcMsgUpdateNamedMsgArrayPtr 800

TpcMsgUpdateNamedMsgPtr 802

TpcMsgUpdateNamedReal4 803

TpcMsgUpdateNamedReal4Array 804

TpcMsgUpdateNamedReal4ArrayPtr 805

TpcMsgUpdateNamedReal8 807

TpcMsgUpdateNamedReal8Array 808

TpcMsgUpdateNamedReal8ArrayPtr 809

TpcMsgUpdateNamedReal16 811

TpcMsgUpdateNamedReal16Array 812

TpcMsgUpdateNamedReal16ArrayPtr 814

TpcMsgUpdateNamedStr 816

TpcMsgUpdateNamedStrArray 817

TpcMsgUpdateNamedStrArrayPtr 818

TpcMsgUpdateNamedStrPtr 820

TpcMsgUpdateNamedTimestamp 821

TpcMsgUpdateNamedTimestampArray 822

TipcMsgUpdateNamedTimestampArrayPtr	823
TipcMsgUpdateNamedUnknown	825
TipcMsgUpdateNamedUtf8	826
TipcMsgUpdateNamedUtf8Array	827
TipcMsgUpdateNamedUtf8ArrayPtr	828
TipcMsgUpdateNamedUtf8Ptr	830
TipcMsgUpdateNamedXml	831
TipcMsgUpdateNamedXmlPtr	832
TipcMsgWrite	833
TipcMsgWriteVa	835
Chapter 11 TipcMt*	839
TipcMtCreate	841
TipcMtDestroy	844
TipcMtGetCompression	845
TipcMtGetDeliveryMode	847
TipcMtGetDeliveryTimeout	850
TipcMtGetGrammar	852
TipcMtGetHeaderStrEncode	853
TipcMtGetLbMode	855
TipcMtGetName	857
TipcMtGetNum	858
TipcMtGetPriority	859
TipcMtGetUserProp	861
TipcMtLogAddMt	863
TipcMtLogRemoveMt	865
TipcMtLookup	866
TipcMtLookupByNum	867
TipcMtPrint	868
TipcMtSetCompression	870
TipcMtSetDeliveryMode	872
TipcMtSetDeliveryTimeout	873
TipcMtSetHeaderStrEncode	874
TipcMtSetLbMode	875
TipcMtSetPriority	876
TipcMtSetPriorityUnknown	877

TipcMtSetUserProp	878
TipcMtTraverse	879

Chapter 12 TipcProperties* 881

TipcPropertiesCreate	882
TipcPropertiesCreateMsg	883
TipcPropertiesDestroy	884
TipcPropertiesGet	885
TipcPropertiesGetCount	886
TipcPropertiesGetDefault	887
TipcPropertiesMsgCreate	888
TipcPropertiesSet	889
TipcPropertiesTraverse	890

Chapter 13 TipcSrv* 891

TipcSrvBufferGetReadSize	895
TipcSrvBufferGetWriteSize	896
TipcSrvCheck	897
TipcSrvCreate	898
TipcSrvCreateCbCreate	904
TipcSrvCreateCbLookup	906
TipcSrvDefaultCbCreate	907
TipcSrvDefaultCbLookup	909
TipcSrvDestroy	911
TipcSrvDestroyCbCreate	915
TipcSrvDestroyCbLookup	918
TipcSrvErrorCbCreate	919
TipcSrvErrorCbLookup	921
TipcSrvFlush	922
TipcSrvGetArch	923
TipcSrvGetAutoFlushSize	925
TipcSrvGetBlockMode	927
TipcSrvGetCurrentConnectedLcn	928
TipcSrvGetConnStatus	929
TipcSrvGetGmdMaxSize	930
TipcSrvGetGmdNumPending	932

TipcSrvGetNode	934
TipcSrvGetNumQueued	936
TipcSrvGetPid	938
TipcSrvGetSocket	939
TipcSrvGetTimeout	940
TipcSrvGetUniqueSubject	942
TipcSrvGetUser	944
TipcSrvGetXtSource	946
TipcSrvGmdFileCreate	948
TipcSrvGmdFileDelete	950
TipcSrvGmdMsgDelete	952
TipcSrvGmdMsgServerDelete	955
TipcSrvGmdMsgStatus	958
TipcSrvIsRunning	963
TipcSrvKeepAlive	964
TipcSrvLock	965
TipcSrvLogAddMt	966
TipcSrvLogRemoveMt	968
TipcSrvMainLoop	969
TipcSrvMsgInsert	971
TipcSrvMsgNext	973
TipcSrvMsgProcess	975
TipcSrvMsgSearch	977
TipcSrvMsgSearchType	979
TipcSrvMsgSend	981
TipcSrvMsgSendRpc	983
TipcSrvMsgWrite	987
TipcSrvMsgWriteVa	989
TipcSrvPrint	991
TipcSrvProcessCbCreate	993
TipcSrvProcessCbLookup	995
TipcSrvQueueCbCreate	997
TipcSrvQueueCbLookup	999
TipcSrvRead	1001
TipcSrvReadCbCreate	1002
TipcSrvReadCbLookup	1004

TipcSrvSetAutoFlushSize	1006
TipcSrvSetBlockMode	1008
TipcSrvSetCredentials	1009
TipcSrvSetGmdMaxSize	1011
TipcSrvSetSocket	1013
TipcSrvSetTimeout	1014
TipcSrvSetUsernamePassword	1016
TipcSrvStdSubjectSetSubscribe	1017
TipcSrvStop	1019
TipcSrvSubjectCbCreate	1021
TipcSrvSubjectCbDestroyAll	1024
TipcSrvSubjectCbLookup	1025
TipcSrvSubjectDefaultCbCreate	1026
TipcSrvSubjectDefaultCbLookup	1028
TipcSrvSubjectGetSubscribe	1029
TipcSrvSubjectGetSubscribeLb	1031
TipcSrvSubjectGmdInit	1033
TipcSrvSubjectLbInit	1035
TipcSrvSubjectSetSubscribe	1037
TipcSrvSubjectSetSubscribeEx	1039
TipcSrvSubjectSetSubscribeLb	1041
TipcSrvSubjectTraverseSubscribe	1043
TipcSrvTrafficGetBytesRecv8	1045
TipcSrvTrafficGetBytesSent8	1046
TipcSrvTrafficGetMsgsRecv8	1047
TipcSrvTrafficGetMsgsSent8	1048
TipcSrvTraverseCbCreate	1049
TipcSrvTraverseCbLookup	1051
TipcSrvUnlock	1052
TipcSrvWriteCbCreate	1053
TipcSrvWriteCbLookup	1055

Chapter 14 TipcSrvConn*	1057
TipcSrvConnBufferGetReadSize	1061
TipcSrvConnBufferGetWriteSize	1062
TipcSrvConnCheck	1063
TipcSrvConnClose	1065
TipcSrvConnCloseCbCreate	1068
TipcSrvConnCloseCbLookup	1070
TipcSrvConnCreate	1071
TipcSrvConnCreateNamed	1072
TipcSrvConnDefaultCbCreate	1073
TipcSrvConnDefaultCbLookup	1075
TipcSrvConnDestroy	1077
TipcSrvConnErrorCbCreate	1078
TipcSrvConnErrorCbLookup	1080
TipcSrvConnFlush	1082
TipcSrvConnGetArch	1083
TipcSrvConnGetAutoFlushSize	1085
TipcSrvConnGetCurrentConnectedLcn	1087
TipcSrvConnGetConnStatus	1088
TipcSrvConnGetGmdDirName	1089
TipcSrvConnGetGmdMaxSize	1090
TipcSrvConnGetGmdNumPending	1092
TipcSrvConnGetNode	1094
TipcSrvConnGetNumQueued	1096
TipcSrvConnGetPid	1098
TipcSrvConnGetProject	1100
TipcSrvConnGetServerNamesList	1101
TipcSrvConnGetSocket	1102
TipcSrvConnGetTimeout	1104
TipcSrvConnGetUniqueSubject	1106
TipcSrvConnGetUser	1108
TipcSrvConnGetXtSource	1110
TipcSrvConnGmdFileCreate	1112
TipcSrvConnGmdFileDelete	1114
TipcSrvConnGmdMsgDelete	1116
TipcSrvConnGmdMsgServerDelete	1119

TipcSrvConnGmdMsgStatus	1123
TipcSrvConnKeepAlive	1128
TipcSrvConnLock	1129
TipcSrvConnMainLoop	1130
TipcSrvConnMsgInsert	1132
TipcSrvConnMsgNext	1134
TipcSrvConnMsgProcess	1136
TipcSrvConnMsgSearch	1138
TipcSrvConnMsgSearchType	1140
TipcSrvConnMsgSend	1142
TipcSrvConnMsgSendRpc	1144
TipcSrvConnMsgWrite	1148
TipcSrvConnMsgWriteVa	1150
TipcSrvConnOpen	1152
TipcSrvConnOpenCbCreate	1154
TipcSrvConnOpenCbLookup	1156
TipcSrvConnPrint	1157
TipcSrvConnProcessCbCreate	1159
TipcSrvConnProcessCbLookup	1161
TipcSrvConnQueueCbCreate	1163
TipcSrvConnQueueCbLookup	1165
TipcSrvConnRead	1167
TipcSrvConnReadCbCreate	1168
TipcSrvConnReadCbLookup	1170
TipcSrvConnSetAutoFlushSize	1172
TipcSrvConnSetCredentials	1174
TipcSrvConnSetGmdDirName	1176
TipcSrvConnSetGmdMaxSize	1177
TipcSrvConnSetProject	1179
TipcSrvConnSetServerNames	1180
TipcSrvConnSetServerNamesList	1181
TipcSrvConnSetTimeout	1182
TipcSrvConnSetUsernamePassword	1184
TipcSrvConnStdSubjectSetSubscribe	1185
TipcSrvConnSubjectCbCreate	1187
TipcSrvConnSubjectCbDestroyAll	1190

TipcSrvConnSubjectCbLookup	1191
TipcSrvConnSubjectDefaultCbCreate	1193
TipcSrvConnSubjectDefaultCbLookup	1195
TipcSrvConnSubjectGetDefaultPrefix	1196
TipcSrvConnSubjectGetSubscribe	1197
TipcSrvConnSubjectGetSubscribeLb	1199
TipcSrvConnSubjectGetUnique	1201
TipcSrvConnSubjectGmdInit	1202
TipcSrvConnSubjectLbInit	1204
TipcSrvConnSubjectSetDefaultPrefix	1206
TipcSrvConnSubjectSetSubscribe	1207
TipcSrvConnSubjectSetSubscribeEx	1209
TipcSrvConnSubjectSetSubscribeLb	1211
TipcSrvConnSubjectSetUnique	1213
TipcSrvConnSubjectTraverseSubscribe	1214
TipcSrvConnTrafficGetBytesRecv8	1216
TipcSrvConnTrafficGetBytesSent8	1217
TipcSrvConnTrafficGetMsgsRecv8	1218
TipcSrvConnTrafficGetMsgsSent8	1219
TipcSrvConnUnlock	1220
TipcSrvConnWriteCbCreate	1221
TipcSrvConnWriteCbLookup	1223
Chapter 15 TipcSrvMon*	1225
TipcSrvMonClientBufferGetWatch	1229
TipcSrvMonClientBufferPoll	1231
TipcSrvMonClientBufferSetWatch	1234
TipcSrvMonClientCbPoll	1238
TipcSrvMonClientCongestionGetWatch	1242
TipcSrvMonClientCongestionSetWatch	1244
TipcSrvMonClientCpuPoll	1248
TipcSrvMonClientExtPoll	1251
TipcSrvMonClientGeneralPoll	1254
TipcSrvMonClientInfoPoll	1259
TipcSrvMonClientMsgRecvGetWatch	1263
TipcSrvMonClientMsgRecvSetWatch	1265

TipcSrvMonClientMsgSendGetWatch	1269
TipcSrvMonClientMsgSendSetWatch	1271
TipcSrvMonClientMsgTrafficPoll	1274
TipcSrvMonClientMsgTypeExPoll	1278
TipcSrvMonClientMsgTypePoll	1284
TipcSrvMonClientNamesGetWatch	1289
TipcSrvMonClientNamesNumPoll	1291
TipcSrvMonClientNamesPoll	1293
TipcSrvMonClientNamesSetWatch	1295
TipcSrvMonClientOptionPoll	1299
TipcSrvMonClientSubjectExPoll	1303
TipcSrvMonClientSubjectPoll	1307
TipcSrvMonClientSubscribeGetWatch	1311
TipcSrvMonClientSubscribeNumPoll	1313
TipcSrvMonClientSubscribePoll	1316
TipcSrvMonClientSubscribeSetWatch	1319
TipcSrvMonClientTimeGetWatch	1323
TipcSrvMonClientTimePoll	1325
TipcSrvMonClientTimeSetWatch	1328
TipcSrvMonClientVersionPoll	1332
TipcSrvMonExtDelete	1334
TipcSrvMonExtSetBinary	1335
TipcSrvMonExtSetBool	1336
TipcSrvMonExtSetBoolArray	1337
TipcSrvMonExtSetInt2	1338
TipcSrvMonExtSetInt2Array	1339
TipcSrvMonExtSetInt4	1340
TipcSrvMonExtSetInt4Array	1341
TipcSrvMonExtSetInt8	1342
TipcSrvMonExtSetInt8Array	1343
TipcSrvMonExtSetReal4	1344
TipcSrvMonExtSetReal4Array	1345
TipcSrvMonExtSetReal8	1346
TipcSrvMonExtSetReal8Array	1347
TipcSrvMonExtSetReal16	1348
TipcSrvMonExtSetReal16Array	1349

TipcSrvMonExtSetStr	1350
TipcSrvMonExtSetStrArray	1351
TipcSrvMonExtSetUtf8	1352
TipcSrvMonExtSetUtf8Array	1353
TipcSrvMonGetIdentStr	1354
TipcSrvMonPrintWatch	1356
TipcSrvMonProjectNamesGetWatch	1358
TipcSrvMonProjectNamesPoll	1360
TipcSrvMonProjectNamesSetWatch	1362
TipcSrvMonServerBufferPoll	1365
TipcSrvMonServerCongestionGetWatch	1369
TipcSrvMonServerCongestionSetWatch	1371
TipcSrvMonServerConnGetWatch	1375
TipcSrvMonServerConnPoll	1377
TipcSrvMonServerConnSetWatch	1380
TipcSrvMonServerCpuPoll	1384
TipcSrvMonServerGeneralPoll	1387
TipcSrvMonServerMaxClientLicensesGetWatch	1392
TipcSrvMonServerMaxClientLicensesSetWatch	1394
TipcSrvMonServerMsgTrafficExPoll	1397
TipcSrvMonServerMsgTrafficPoll	1401
TipcSrvMonServerNamesGetWatch	1405
TipcSrvMonServerNamesPoll	1407
TipcSrvMonServerNamesSetWatch	1409
TipcSrvMonServerOptionPoll	1412
TipcSrvMonServerRoutePoll	1416
TipcSrvMonServerStartTimePoll	1420
TipcSrvMonServerTimePoll	1423
TipcSrvMonServerVersionPoll	1426
TipcSrvMonSetIdentStr	1428
TipcSrvMonSubjectNamesGetWatch	1429
TipcSrvMonSubjectNamesPoll	1431
TipcSrvMonSubjectNamesSetWatch	1433
TipcSrvMonSubjectSubscribeGetWatch	1436
TipcSrvMonSubjectSubscribePoll	1438
TipcSrvMonSubjectSubscribeSetWatch	1441

Chapter 16 TipcStrToDeliveryMode - TipcSubjectValid 1445

TipcStrToDeliveryMode 1446

TipcStrToFt 1448

TipcStrToLbMode 1450

TipcSubjectGetUnique 1451

TipcSubjectMatch 1452

TipcSubjectValid 1453

Index 1455

Preface

TIBCO SmartSockets is a message-oriented middleware product that enables programs to communicate quickly, reliably, and securely across:

- local area networks (LANs)
- wide area networks (WANs)
- the Internet

SmartSockets takes care of network interfaces, guarantees delivery of messages, handles communications protocols, and directs recovery after system or network problems. This enables you to focus on higher-level requirements rather than the underlying complexities of the network.

This reference provides the detailed information you need to understand and use the SmartSockets system. Before using this reference, it is helpful to read the *TIBCO SmartSockets Tutorial* and work through the lessons in that book. The *TIBCO SmartSockets Tutorial* begins at a more basic level and is an excellent introduction to SmartSockets.

For an overview of the new features, changes, and enhancements in this Version 6.8 release, see the *TIBCO SmartSockets Installation Guide*.

Topics

- *Related Documentation, page xxviii*
- *Conventions Used in This Manual, page xxix*
- *How to Contact TIBCO Support, page xxxii*

Related Documentation

This section lists documentation resources you may find useful.

TIBCO Product Documentation

The following documents form the TIBCO SmartSockets documentation set:

- *TIBCO SmartSockets API Quick Reference*
- *TIBCO SmartSockets Application Programming Interface*
- *TIBCO SmartSockets C++ User's Guide*
- *TIBCO SmartSockets cxxipc Class Library*
- *TIBCO SmartSockets Installation Guide*
- *TIBCO SmartSockets Java Library User's Guide and Tutorial*
- *TIBCO SmartSockets .NET User's Guide and Tutorial*
- *TIBCO SmartSockets Tutorial*
- *TIBCO SmartSockets User's Guide*
- *TIBCO SmartSockets Utilities*
- *TIBCO SmartSockets C++ and Java Class Libraries*

C++ class library and Java application programming interface (API) materials are available in HTML format only. Access the references through the TIBCO HTML documentation interface.

Using the Online Documentation

The SmartSockets documentation files are available for you to download separately, or you can request a copy of the TIBCO Documentation CD.

Conventions Used in This Manual

This manual uses the following conventions.

Typeface Conventions

This manual uses the following typeface conventions

Example	Use
<code>monospace</code>	This monospace font is used for program output and code example listing and for file names, commands, configuration file parameters, and literal programming elements in running text.
<code>monospace bold</code>	This bold monospace font indicates characters in a command line that you must type exactly as shown. This font is also used for emphasis in code examples.
<i>Italic</i>	Italic text is used as follows: • In code examples, file names etc., for text that should be replaced with an actual value. For example: "Select <i>install-dir</i>/runexample.bat." • For document titles. • For emphasis.
Bold	Bold text indicates actions you take when using a GUI, for example, click OK, or choose Edit from the menu. It is intended to help you skim through procedures when you are familiar with them and just want a reminder. Submenus and options of a menu item are indicated with an angle bracket, for example, Menu > Submenu.
	Warning. The accompanying text describes a condition that severely affects the functioning of the software.
	Note. Be sure you read the accompanying text for important information.
	Tip. The accompanying text may be especially helpful.

Notational Conventions

The notational conventions in the table below are used for describing command syntax. When used in this context, do not type the brackets listed in the table as part of a command line.

Notation	Description	Use
[]	Brackets	Used to enclose an optional item in the command syntax.
< >	Angle Brackets	Used to enclose a name (usually in <i>Italic</i>) that represents an argument for which you substitute a value when you use the command. This convention is not used for XML or HTML examples or other situations where the angle brackets are part of the code.
{ }	Curly Brackets	Used to enclose two or more items among which you can choose only one at a time. Vertical bars () separate the choices within the curly brackets.
...	Ellipsis	Indicates that you can repeat an item any number of times in the command line.

Identifiers

The term identifier is used to refer to a valid character string that names entities created in a SmartSockets application. The string starts with an underscore (`_`) or alphabetic character and is followed by zero or more letters, digits, percent signs (`%`), or underscores. No other special characters are valid. The maximum length of the string is 63 characters. Identifiers are not case-sensitive.

These are examples of valid identifiers:

```
EPS
battery_11
K11
__
_all
```

These are invalid identifiers:

```
20
battery-11
@com
$amount
```

Case

Function names are case-sensitive, and must use the mixed-case format you see in the text. For example, `TipcMsgCreate`, `TipcSrvStop`, and `TipcMonClientMsgTrafficPoll` are SmartSockets functions and must use the case as shown.

Monitoring messages are also case-sensitive, and should be all upper case, such as `T_MT_MON_SERVER_NAMES_POLL_CALL`. This makes it easy to distinguish them from option or function names.

Although option names are not case-sensitive, they are usually presented in text with mixed case, to help distinguish them from commands or other items. For example, `Server_Names`, `Unique_Subject`, and `Project` are all SmartSockets options.

Identifiers used with the products in the SmartSockets family are not case-sensitive. For example, the identifiers `thermal` and `THERMAL` are equivalent in all processes.

In UNIX, shell commands and filenames are case-sensitive, though they might not be in other operating systems, such as Windows. To make it easier to port applications between operating systems, always specify filenames in lower case.

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, please contact TIBCO Support as follows.

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

<http://www.tibco.com/services/support>

- If you already have a valid maintenance or support contract, visit this site:

<http://support.tibco.com>

Entry to this site requires a user name and password. If you do not have a user name, you can request one.

Chapter 1 **Introduction**

The TIBCO SmartSockets interprocess communications (IPC) Application Programming Interface (API) allows you to send and receive messages asynchronously across a heterogeneous network.

Topics

- *Overview, page 2*
- *Naming Conventions, page 3*
- *Essential API Functions, page 4*
- *Data Structures, page 6*
- *Include Files, page 7*
- *Compiling and Linking, page 8*
- *Messages, page 10*
- *Message Types, page 12*
- *Message Files, page 13*
- *Connections, page 13*
- *Global Connections, page 17*
- *Multiple Connections, page 20*
- *Properties Table, page 21*
- *Error Handling, page 22*

Overview

The TIBCO SmartSockets interprocess communications (IPC) Application Programming Interface (API) allows you to send and receive messages asynchronously across a heterogeneous network.

The IPC API can, among other things, be used to:

- send and receive prioritized messages between two processes through connections
- send messages with or without guaranteed message delivery (GMD)
- execute user-defined callback functions when certain operations occur
- use subjects to send one message to many processes with RTserver (publish-subscribe)
- use connections and publish-subscribe from multi-threaded programs
- detect network failures and initiate actions to recover from them, such as restarting processes, switching over from primary to backup processes, and retransmitting messages to guarantee delivery
- send messages using different IPC protocols over a heterogeneous network of computers
- monitor processes by watching and polling for IPC-related information such as projects in use, subjects in use, messages sent and received, names of all RTclients and RTservers, and so on

In general, these API functions can be called from any program. Typically, programs are written in C or C++. However, any language that supports a C binding can effectively use the SmartSockets API.

In addition to the C-based API described in this chapter, other class libraries are included in SmartSockets:

- A C++ class library is included with TIBCO SmartSockets and is described in the *TIBCO SmartSockets C++ User's Guide*.
- A Java class library is included with TIBCO SmartSockets and is described in the *TIBCO SmartSockets Java Library User's Guide and Tutorial*.

Naming Conventions

The SmartSockets API uses these naming conventions:

- All IPC functions start with `Tipc`.
- All SmartSockets message type numbers start with `T_MT_`. In the text, message types are frequently referred to without the `T_MT_` prefix. In code however, they are always referred to with their full names.
- Most IPC typedefs, enumerated values, and defines start with `T_IPC_`.
- All message functions start with `TipcMsg`.
- All message type functions start with `TipcMt`.
- All connection functions start with `TipcConn`.
- All functions that communicate with RTserver using global connections start with `TipcSrv`.
- All functions that communicate with RTserver using multiple connections start with `TipcSrvConn`.
- All monitoring functions using global connections start with `TipcMon`.
- All monitoring functions using multiple connections start with `TipcSrvMon`.

Essential API Functions

This API covers a wide range of functionality. While all of this functionality is available to a SmartSockets application developer, the typical SmartSockets application uses only a small fraction of the available functions. This list covers the functions that every SmartSocket application will use.

Message Type Functions

These functions create and retrieve information about message types:

- **TipcMtCreate** — Create a new message type.
- **TipcMtLookup** — Look up a message type by name. This is necessary to create a new message.

Message Functions

These functions construct, manipulate and destroy messages:

- **TipcMsgCreate** — Create a message.
- **TipcMsgSetDest** — Set the destination, typically the subject name to which this message will be published.
- **TipcMsgAppend*** — This is a class of functions to append various types of fields to a message, including binaries, strings, real numbers, and so on.
- **TipcSrvMsgSend** and **TipcSrvConnMsgSend** — Publish a message.
- **TipcMsgSetCurrent** — Set the current field of a message.
- **TipcMsgNext*** (**TipcMsgNextBinary** through **TipcMsgNextXml**) — This is a class of functions that reads the next field of the type specified. Analogous to the **TipcMsgAppend*** functions, there is a "get next" for binaries, strings, real numbers, and so on.
- **TipcMsgGetNamed*** (**TipcMsgGetNamedBinary** through **TipcMsgGetNamedXml**) — This is a class of functions that gets a field by name. Analogous to the **TipcMsgAddNamed*** functions, there is a "get named" for binaries, strings, reals, and other types.
- **TipcMsgDestroy** — Destroy a message.

Communication Functions

These are RTclient functions that communicate with RTserver to receive and process messages:

- `TipcSrvSubjectCbCreate` and `TipcSrvConnSubjectCbCreate` — Define a new callback to be invoked when a message is received to a particular subject.
- `TipcSrvSubjectDefaultCbCreate` and `TipcSrvConnSubjectDefaultCbCreate` — Define a default subject callback, to be invoked when no other subject callback is defined for that subject.
- `TipcSrvSubjectSetSubscribe` and `TipcSrvConnSubjectSetSubscribe`— Start or stop subscribing to a subject.
- `TipcSrvMainLoop` and `TipcSrvConnMainLoop` — Read and process messages from RTserver.

Utility Functions

These functions are documented in *TIBCO SmartSockets Utilities*, and provide help with startup, debugging and clean up:

- `TutCommandParseFile` — Parse commands from a `.cmd` file for startup configuration.
- `TutErrNumGet` — Get the value of the global SmartSockets error number.
- `TutErrStrGet` — Get the SmartSockets global error number as a descriptive string.
- `TutOut` — Unconditionally print out to a SmartSockets output window.
- `TutExit` — Call exit handlers and terminate the process. This is required in Windows applications.

Data Structures

There are a number of SmartSockets IPC data structures accessible from user-written programs. This is how these structures are defined in `<rtworks/ipc.h>`:

```
typedef struct T_IPC_MT_STRUCT T_IPC_MT_STRUCT, *T_IPC_MT;  
typedef struct T_IPC_MSG_STRUCT T_IPC_MSG_STRUCT, *T_IPC_MSG;  
typedef struct T_IPC_MSG_FIELD_STRUCT T_IPC_MSG_FIELD_STRUCT,  
    *T_IPC_MSG_FIELD;  
typedef struct T_IPC_MSG_FILE_STRUCT T_IPC_MSG_FILE_STRUCT,  
    *T_IPC_MSG_FILE;  
typedef struct T_IPC_CONN_STRUCT T_IPC_CONN_STRUCT, *T_IPC_CONN;
```

These structure definitions are incomplete in the sense that none of the fields in the structure are defined here. This is one way of creating an opaque data type in C. Pointers to these structures cannot be dereferenced to look at what they point to. These pointers can only be manipulated through access functions. This protects users from changes to the underlying data structures.

Include Files

Several important header files are needed to compile and link with SmartSockets. These files are normally included in C/C++ code with the `#include C/C++` preprocessor directive. The TIBCO SmartSockets header files are located in these directories:

UNIX:

```
$RTHOME/include/$RTARCH/rtworks
```

OpenVMS:

```
RTHOME: [ INCLUDE . RTWORKS ]
```

Windows:

```
%RTHOME%\include\rtworks
```

Include header files with the angle-bracket syntax. For example, when using C, this inclusion line is used:

```
#include <rtworks/ipc.h>
```

Use of the angle-bracket syntax is preferred over the use of double-quotes, as shown here:

```
#include "ipc.h"
```

or

```
#include "pathname/ipc.h"
```

On UNIX, using angle brackets requires using an argument to the C++ command.

```
$ CC -I$RTHOME/include/$RTARCH ...
```

On OpenVMS, the logical RTWORKS directory is `RTHOME: [ INCLUDE . RTWORKS ]` so that DEC C++ uses that directory for `<rtworks/ipc.h>`.

On Windows, add the SmartSockets include directory `%RTHOME%\include\rtworks` to the list of Visual C++ Developer Studio project include directories. There are several example Visual C++ makefiles in `%RTHOME%\examples\smrtsock\manual` that you can use as a starting point for your application executables.

Compiling and Linking

After writing a user-defined program, use the `rtlink` shell script to link it with the necessary TIBCO SmartSockets libraries. These sections present some examples of using the `rtlink` script to link the program stored in the `myprog.c` and `extfunc.c` files.

On UNIX

Here are examples of two methods of invoking the `rtlink` script on UNIX. In one example, the compiling and linking steps are executed separately; in the other, they are combined into one step. Both examples use the `-o` flag to explicitly name the executable, and the `-O` flag to enable compiler and SmartSockets optimization. By default, `rtlink` uses the `cc` command to compile and link, but you can override this by using the `CC` environment variable. For more information on compiling and linking, see the section on `rtlink` in *TIBCO SmartSockets Utilities*.

```
/* Separate compiles followed by a separate link */
$ cc -O -I$RTHOME/include/$RTARCH -c myprog.c
$ cc -O -I$RTHOME/include/$RTARCH -c extfunc.c
$ rtlink -O myprog.o extfunc.o -o myprog.x
```

```
/* Combined compiles and link */
$ rtlink -O myprog.c extfunc.c -o myprog.x
```

Using a Different Compiler

By default, `rtlink` uses the `cc` command to compile and link files. If the `CC` environment variable is set, `rtlink` instead uses the `CC` environment variable to determine the command to invoke. To use a different compiler, set this environment variable to the compiler you wish to use. Here is an example using a C++ compiler:

```
$ env CC=g++ rtlink -O myprog.C extfunc.C -o myprog.x
```



Never set `CC` to `rtlink`. This causes recursion, as `rtlink` calls itself over and over.

Using Make

If you are using the make utility to control and execute your compilations, you can instead set the CC variable in your makefile and make will automatically set the CC environment variable each time it runs a command. Here is an example of setting CC from a makefile to use the GNU C compiler:

```
CC = gcc
CFLAGS = -O -I$(RTHOME)/include/$(RTARCH)
OBJECTS = myprog.o extfunc.o
```

```
myprog.x: $(OBJECTS)
    rtlink $(OBJECTS) -o $@
```

If you are using GNU make, this may be needed instead in your makefile or GNUmakefile:

```
export CC = gcc
CFLAGS = -O -I$(RTHOME)/include/$(RTARCH)
OBJECTS = myprog.o extfunc.o
```

```
myprog.x: $(OBJECTS)
    rtlink $(OBJECTS) -o $@
```

On OpenVMS

On OpenVMS, the `rtlink` syntax is similar to that for UNIX, except that it is not possible to combine the compilation and link steps. These steps must be explicitly invoked. This example uses the `/executable` qualifier to explicitly name the executable. For more information on compiling and linking, see the section on `rtlink` in *TIBCO SmartSockets Utilities*.

```
/* Separate compiles followed by a separate link */
$ cc/optimize myprog.c
$ cc/optimize extfunc.c
$ rtlink/executable=myprog.exe myprog.obj extfunc.obj
```

The compilations create object files named `myprog.obj` and `extfunc.obj`. `rtlink` creates an executable named `myprog.exe`.

On Windows

On Windows, compiling and linking is usually done in the Visual C++ Developer Studio environment. There is no `rtlink` script on Windows. You can use one of several example Visual C++ makefiles in `%RTHOME%\examples\smrtsock\manual` as a starting point for your application executables.

Under Windows, if you are using SmartSockets for the first time, you need to set the paths for your include files and libraries in Developer Studio. You must also set the Project settings in Developer Studio. See the *TIBCO SmartSockets Installation Guide* for details on this.

Messages

Messages are packets of information sent from one process to one or more other processes providing instructions or data for the receiving process. Messages are described in detail in the *TIBCO SmartSockets User's Guide*.

The fields in a message can be traversed with the function `TipcMsgTraverse`. It uses a traversal function defined in this way:

```
typedef T_PTR (*T_IPC_MSG_TRAV_FUNC)
(T_IPC_MSG msg, T_IPC_MSG_TRAV field, T_PTR arg);
```

For more information on traversal functions, see *TIBCO SmartSockets Utilities*. The specified traversal function for `TipcMsgTraverse` must be defined with these arguments. The second argument to the traversal function contains information for a field in a message. This is how the data structure for this argument is defined:

```
typedef struct T_IPC_MSG_TRAV_STRUCT T_IPC_MSG_TRAV_STRUCT,
*T_IPC_MSG_TRAV;
struct T_IPC_MSG_TRAV_STRUCT {
    T_INT2 *type_ptr; /* stored as 2 bytes in message */
    T_PTR value_ptr; /* can be cast to appropriate type */
    T_INT4 *array_size_ptr; /* NULL if not applicable */
    T_IPC_MSG_FIELD field; /* actual field */
};
```

The field type in a message is stored as a 2-byte integer (to save space), but should be referenced in code with the enumerated type `T_IPC_FT`. This is how `T_IPC_FT` is defined:

```
typedef enum {
    T_IPC_FT_INVALID,
    T_IPC_FT_CHAR,
    T_IPC_FT_BINARY,
    T_IPC_FT_STR,
    T_IPC_FT_STR_ARRAY,
    T_IPC_FT_INT2,
    T_IPC_FT_INT2_ARRAY,
    T_IPC_FT_INT4,
    T_IPC_FT_INT4_ARRAY,
    T_IPC_FT_INT8,
    T_IPC_FT_INT8_ARRAY,
    T_IPC_FT_REAL4,
    T_IPC_FT_REAL4_ARRAY,
    T_IPC_FT_REAL8,
    T_IPC_FT_REAL8_ARRAY,
    T_IPC_FT_REAL16,
    T_IPC_FT_REAL16_ARRAY,
    T_IPC_FT_MSG,
    T_IPC_FT_MSG_ARRAY,
    T_IPC_FT_INTERNAL_GROUP, /* used internally for message files */
    T_IPC_FT_INTERNAL_BOOL, /* used internally for message files */
    T_IPC_FT_INTERNAL_ID, /* used internally for message files */
    T_IPC_FT_INTERNAL_VERBOSE, /* used internally for message files */
    T_IPC_PROP_DELIVERY_MODE, /* access a property */
    T_IPC_PROP_DEST, /* access a property */
    T_IPC_PROP_PRIORITY, /* access a property */
    T_IPC_PROP_SENDER, /* access a property */
    T_IPC_PROP_TYPE, /* access a property */
    T_IPC_PROP_USER_PROP, /* access a property */
    T_IPC_PROP_DEST_SERVER, /* undocumented */
    T_IPC_PROP_SCOPE, /* undocumented */
    T_IPC_PROP_DELIVERY_TIMEOUT, /* access a property */
    T_IPC_PROP_LB_MODE, /* access a property */
    T_IPC_PROP_HEADER_STR_ENCODE /* access a property */
} T_IPC_FT;
```

The `T_IPC_PROP_*` enumerated values are not field types, but instead allow a message property to be accessed with the convenience functions `TipcMsgRead`, `TipcMsgReadVa`, `TipcMsgWrite`, `TipcMsgWriteVa`, `TipcConnMsgWrite`, `TipcConnMsgWriteVa`, `TipcSrvMsgWrite`, and `TipcSrvMsgWriteVa`.

Both messages and message types (see Message Typespage 12) have a delivery mode property that controls what level of guarantee is used when a message is sent through a connection with `TipcConnMsgSend`. This is how the enumerated type used to identify this property is defined:

```
typedef enum {
    T_IPC_DELIVERY_BEST_EFFORT, /* network failure can cause loss */
    T_IPC_DELIVERY_SOME,       /* guaranteed to deliver to at least one */
    T_IPC_DELIVERY_ALL,        /* guaranteed to deliver to all receivers */
    T_IPC_DELIVERY_ORDERED     /* despite failure, delivers in order published */
} T_IPC_DELIVERY_MODE;
```

Both messages and message types (see Message Typespage 12) have a load balancing mode property that controls how a publish-subscribe message is load balanced. The enumerated type used to identify this property is defined as:

```
typedef enum {
    T_IPC_LB_NONE, /* normal publish subscribe delivery mode */
    T_IPC_LB_ROUND_ROBIN, /* Round Robin load balancing */
    T_IPC_LB_WEIGHTED, /* weighted load balancing (based on GMD acks) */
    T_IPC_LB_SORTED /* sorted load balancing */
} T_IPC_LB_MODE;
```

Message Types

Each message has a type property that defines the purpose of the message. Message types are described in detail in the *TIBCO SmartSockets User's Guide*. All of the standard message types that SmartSockets supports are defined in `<rtworks/mt.h>`, which is included by `<rtworks/ipc.h>`. These standard message types are also listed in the *TIBCO SmartSockets User's Guide*.

The list of known message types can be traversed with the function `TipcMtTraverse`. It uses this traversal function:

```
typedef T_PTR (*T_IPC_MT_TRAV_FUNC)
(T_STR name, T_IPC_MT mt, T_PTR arg);
```

For more information on traversal functions, see *TIBCO SmartSockets Utilities*. The specified traversal function for `TipcMtTraverse` must be defined with these arguments.

Message Files

A message file is a text or binary file containing one or more messages. Message files are described in detail in the *TIBCO SmartSockets User's Guide*. A message file is created with the function `TipcMsgFileCreate` or `TipcMsgFileCreateFromFile` using a creation mode that is defined in this way:

```
typedef enum {
    T_IPC_MSG_FILE_CREATE_READ, /* read (auto-detect text or binary) */
    T_IPC_MSG_FILE_CREATE_WRITE, /* write in text mode */
    T_IPC_MSG_FILE_CREATE_WRITE_BINARY, /* write in binary mode */
    T_IPC_MSG_FILE_CREATE_APPEND /* append (auto-detect text or binary) */
} T_IPC_MSG_FILE_CREATE_MODE;
```

Connections

A connection is an endpoint of a communication link used to send and receive messages between two processes. Connections are described in detail in the *TIBCO SmartSockets User's Guide*.

A connection has several timeout properties that are used to check for possible network failures. These timeouts are accessed with the functions `TipcConnGetTimeout` and `TipcConnSetTimeout`. This is how the enumerated type used to identify these properties is defined:

```
typedef enum {
    T_IPC_TIMEOUT_READ,
    T_IPC_TIMEOUT_WRITE,
    T_IPC_TIMEOUT_KEEP_ALIVE,
    T_IPC_TIMEOUT_DELIVERY
} T_IPC_TIMEOUT;
```

The function `TipcConnMsgSearch` can be used to search a connection's message queue for a specific message. It uses a search function defined in this way:

```
typedef T_BOOL (*T_IPC_CONN_MSG_SEARCH_FUNC)
(T_IPC_CONN conn, T_IPC_MSG msg, T_PTR arg);
```

Connection Callbacks

Connections have several types of callbacks. For more information on connection callbacks, see the *TIBCO SmartSockets User's Guide*. Each callback type has a callback function type and a callback data type.

Process, Default, Read, and Write Callbacks

The connection process, default, read, and write callbacks all use similar data structures. This is how the data structures are defined:

```
typedef void T_ENTRY (*T_IPC_CONN_MSG_CB_FUNC)
    (T_IPC_CONN conn, T_IPC_CONN_MSG_CB_DATA data, T_CB_ARG arg);

typedef T_IPC_CONN_MSG_CB_FUNC T_IPC_CONN_PROCESS_CB_FUNC;

typedef T_IPC_CONN_MSG_CB_FUNC T_IPC_CONN_DEFAULT_CB_FUNC;

typedef T_IPC_CONN_MSG_CB_FUNC T_IPC_CONN_WRITE_CB_FUNC;

typedef void T_ENTRY (*T_IPC_CONN_READ_CB_FUNC)
    (T_IPC_CONN conn, T_IPC_CONN_READ_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_CONN_MSG_CB_DATA_STRUCT
    T_IPC_CONN_MSG_CB_DATA_STRUCT, *T_IPC_CONN_MSG_CB_DATA;
struct T_IPC_CONN_MSG_CB_DATA_STRUCT {
    T_CB cb;
    T_IPC_MSG msg;
};

typedef T_IPC_CONN_MSG_CB_DATA_STRUCT
T_IPC_CONN_WRITE_CB_DATA_STRUCT,
    *T_IPC_CONN_WRITE_CB_DATA;

typedef T_IPC_CONN_MSG_CB_DATA_STRUCT
    T_IPC_CONN_PROCESS_CB_DATA_STRUCT, *T_IPC_CONN_PROCESS_CB_DATA;

typedef T_IPC_CONN_MSG_CB_DATA_STRUCT
    T_IPC_CONN_DEFAULT_CB_DATA_STRUCT, *T_IPC_CONN_DEFAULT_CB_DATA;

typedef struct T_IPC_CONN_READ_CB_DATA_STRUCT
    T_IPC_CONN_READ_CB_DATA_STRUCT, *T_IPC_CONN_READ_CB_DATA;
struct T_IPC_CONN_READ_CB_DATA_STRUCT {
    T_CB cb;
    T_IPC_MSG msg;
    T_INT4 packet_size; /* make available for monitoring */
};
```

Error Callbacks

This is how the connection error callback data structures are defined:

```
typedef void T_ENTRY (*T_IPC_CONN_ERROR_CB_FUNC)
    (T_IPC_CONN conn, T_IPC_CONN_ERROR_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_CONN_ERROR_CB_DATA_STRUCT
    T_IPC_CONN_ERROR_CB_DATA_STRUCT, *T_IPC_CONN_ERROR_CB_DATA;
struct T_IPC_CONN_ERROR_CB_DATA_STRUCT {
    T_CB cb;
    T_INT4 c_err_num; /* C error number (errno) */
    T_INT4 os_err_num; /* OS error number (vaxc$errno, GetLastError(), etc.) */
    T_INT4 socket_err_num; /* socket error number (WSAGetLastError(), */
    /* sock_errno(), etc.) */
};
```

Queue Callbacks

This is how the connection queue callback data structures are defined:

```
typedef void T_ENTRY (*T_IPC_CONN_QUEUE_CB_FUNC)
    (T_IPC_CONN conn, T_IPC_CONN_QUEUE_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_CONN_QUEUE_CB_DATA_STRUCT
    T_IPC_CONN_QUEUE_CB_DATA_STRUCT, *T_IPC_CONN_QUEUE_CB_DATA;
struct T_IPC_CONN_QUEUE_CB_DATA_STRUCT {
    T_CB cb;
    T_IPC_MSG msg; /* message inserted into or deleted from queue */
    T_BOOL insert_flag; /* TRUE if inserted, FALSE if deleted */
    T_INT4 position; /* position of message in queue (0 is front) */
};
```

Encode and Decode Callbacks

This is how the connection encode and decode callback data structures are defined:

```
typedef void T_ENTRY (*T_IPC_CONN_ENCODE_CB_FUNC)
    (T_IPC_CONN conn, T_IPC_CONN_ENCODE_CB_DATA data, T_CB_ARG arg);

typedef void T_ENTRY (*T_IPC_CONN_DECODE_CB_FUNC)
    (T_IPC_CONN conn, T_IPC_CONN_DECODE_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_CONN_ENCODE_CB_DATA_STRUCT
    T_IPC_CONN_ENCODE_CB_DATA_STRUCT,
    *T_IPC_CONN_ENCODE_CB_DATA;
struct T_IPC_CONN_ENCODE_CB_DATA_STRUCT {
    T_CB cb;
    T_PTR packet; /* packet minus counter */
    T_INT4 packet_size; /* number of bytes in packet */
    T_PTR orig_packet; /* for reference */
    T_INT4 orig_packet_size; /* for reference */
    T_IPC_MSG orig_msg; /* make available for monitoring */
};

typedef struct T_IPC_CONN_DECODE_CB_DATA_STRUCT
    T_IPC_CONN_DECODE_CB_DATA_STRUCT,
    *T_IPC_CONN_DECODE_CB_DATA;
struct T_IPC_CONN_DECODE_CB_DATA_STRUCT {
    T_CB cb;
    T_PTR packet; /* packet minus counter */
    T_INT4 packet_size; /* number of bytes in packet */
    T_PTR orig_packet; /* for reference */
    T_INT4 orig_packet_size; /* for reference */
};
```

Global Connections

RTserver and RTclient allow many processes to communicate easily with each other. RTserver and RTclient are discussed in detail in the *TIBCO SmartSockets User's Guide*. Most RTclient applications require only a single connection to an RTserver. This single global connection is created and managed using the TpcSrv API. RTclient applications that require multiple connections to RTservers use the TpcSrvConn API, covered in Multiple Connections page 20.

The functions TpcSrvCreate, TpcSrvDestroy, and TpcSrvGetConnStatus manipulate the status of the connection to RTserver. These functions use an enumerated type; this is how the enumerated type is defined:

```
typedef enum {
    T_IPC_SRV_CONN_NONE, /* not connected at all */
    T_IPC_SRV_CONN_WARM, /* warm connection to RTserver */
    T_IPC_SRV_CONN_FULL /* fully connected to RTserver */
} T_IPC_SRV_CONN_STATUS;
```

The functions TpcSrvLogAddMt and TpcSrvLogRemoveMt add a message type to or remove a message type from a message file logging type. These functions use an enumerated type. This is how the enumerated type is defined:

```
typedef enum {
    T_IPC_SRV_LOG_DATA, /* slot values and time */
    T_IPC_SRV_LOG_STATUS, /* status messages */
    T_IPC_SRV_LOG_INTERNAL /* all other standard message types */
} T_IPC_SRV_LOG_TYPE;
```

The function TpcSrvStop stops RTserver and possibly more processes, using an enumerated type. This is how the enumerated type is defined:

```
typedef enum {
    T_IPC_SRV_STOP_NORMAL = 1, /* just stop one RTserver */
    T_IPC_SRV_STOP_ALL, /* stop all RTservers and RTclients */
    T_IPC_SRV_STOP_CLIENTS, /* stop one RTserver and its RTclients */
    T_IPC_SRV_STOP_SERVERS /* stop all RTservers */
} T_IPC_SRV_STOP_TYPE;
```

The function TpcSrvSubjectTraverseSubscribe can be used to traverse the subjects that an RTclient process is subscribing to. This is how the traversal function is defined:

```
typedef T_PTR (*T_IPC_SRV_SUBJECT_TRAV_FUNC)
(T_STR name, T_STR not_used, T_PTR arg);
```

Global Connection Callbacks

In addition to connection callbacks, an RTclient process can have server create callbacks, server destroy callbacks, server names traverse callbacks, and subject callbacks. (See Connection Callbacks page 14 for information on connection callbacks.) Each callback type has a callback function type and callback data type.

Server Create Callbacks

This is how the server create callback data structures are defined:

```
typedef void T_ENTRY (*T_IPC_SRV_CREATE_CB_FUNC)
    (T_IPC_CONN conn, T_IPC_SRV_CREATE_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_SRV_CREATE_CB_DATA_STRUCT
    T_IPC_SRV_CREATE_CB_DATA_STRUCT, *T_IPC_SRV_CREATE_CB_DATA;
struct T_IPC_SRV_CREATE_CB_DATA_STRUCT {
    T_CB cb;
    T_IPC_SRV_CONN_STATUS old_conn_status; /* old connection status */
    T_IPC_SRV_CONN_STATUS new_conn_status; /* new connection status */
};
```

Server Destroy Callbacks

This is how the server destroy callback data structures are defined:

```
typedef void T_ENTRY (*T_IPC_SRV_DESTROY_CB_FUNC)
    (T_IPC_CONN conn, T_IPC_SRV_DESTROY_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_SRV_DESTROY_CB_DATA_STRUCT
    T_IPC_SRV_DESTROY_CB_DATA_STRUCT, *T_IPC_SRV_DESTROY_CB_DATA;
struct T_IPC_SRV_DESTROY_CB_DATA_STRUCT {
    T_CB cb;
    T_IPC_SRV_CONN_STATUS old_conn_status; /* old connection status */
    T_IPC_SRV_CONN_STATUS new_conn_status; /* new connection status */
};
```

Server Names Traverse Callbacks

This is how the server traverse callback data structures are defined:

```
typedef void (T_ENTRY1 * T_ENTRY2 T_EXPORT
T_IPC_SRV_TRAVERSE_CB_FUNC)
T_TYPEDEF((T_IPC_CONN,          /* conn */
          T_IPC_SRV_TRAVERSE_CB_DATA, /* data */
          T_CB_ARG));          /* arg */

typedef struct T_IPC_SRV_TRAVERSE_CB_DATA_STRUCT
          T_IPC_SRV_TRAVERSE_CB_DATA_STRUCT,
          *T_IPC_SRV_TRAVERSE_CB_DATA;

struct T_IPC_SRV_TRAVERSE_CB_DATA_STRUCT {
    T_CB cb;
    T_STR server_name;      /* next server name in the list */
    T_BOOL stop_traverse;   /* set by callback to stop traversing */
};
```

Subject Callbacks

This is how the server subject callback data structures are defined:

```
typedef void T_ENTRY (*T_IPC_CONN_MSG_CB_FUNC)
          (T_IPC_CONN conn, T_IPC_CONN_MSG_CB_DATA data, T_CB_ARG arg);

typedef T_IPC_CONN_MSG_CB_FUNC T_IPC_SRV_SUBJECT_CB_FUNC;

struct T_IPC_CONN_MSG_CB_DATA_STRUCT {
    T_CB cb;
    T_IPC_MSG msg;
};

typedef T_IPC_CONN_MSG_CB_DATA_STRUCT
          T_IPC_SRV_SUBJECT_CB_DATA_STRUCT, *T_IPC_SRV_SUBJECT_CB_DATA;
```

Multiple Connections

For those rare applications whose architecture requires an RTclient to have connections to more than one RTserver, those RTclients can use multiple connections instead of a single global connection. The multiple connections use the TipcSrvConn API. The TipcSrvConn API allows distinct connections to multiple RTservers with each connection having its own set of subscriptions and callbacks. You should not mix types of connections within an RTclient. An RTclient should connect to RTservers either with a single global connection or with multiple connections. For more information on the enumerated types and callbacks used by global connections, see Global Connections page 17.

Multiple Connection Callbacks

An RTclient process can have server create callbacks and server close callbacks. Each callback type has a callback function type and callback data type.

Server Open Callbacks

When using multiple connections, the server open callback may be used. This is how the server open callback data structures are defined:

```
typedef void (*T_IPC_SRV_OPEN_CB_FUNC)
    (T_IPC_SRV srv, T_IPC_SRV_OPEN_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_SRV_OPEN_CB_DATA_STRUCT
    T_IPC_SRV_OPEN_CB_DATA_STRUCT, *T_IPC_SRV_OPEN_CB_DATA;
struct T_IPC_SRV_OPEN_CB_DATA_STRUCT {
    T_CB cb;
    T_IPC_SRV_CONN_STATUS old_conn_status;
    T_IPC_SRV_CONN_STATUS new_conn_status;
};
```

Server Close Callbacks

When using multiple connections, the server close callback may be used. This is how the server close callback data structures are defined:

```
typedef void (*T_IPC_SRV_CLOSE_CB_FUNC)
    (T_IPC_SRV srv, T_IPC_SRV_CLOSE_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_SRV_CLOSE_CB_DATA_STRUCT
    T_IPC_SRV_CLOSE_CB_DATA_STRUCT, *T_IPC_SRV_CLOSE_CB_DATA;
struct T_IPC_SRV_CLOSE_CB_DATA_STRUCT {
    T_CB cb;
    T_IPC_SRV_CONN_STATUS old_conn_status;
    T_IPC_SRV_CONN_STATUS new_conn_status;
};
```


Events

Each event has an associated dispatch function defined as:

```
typedef void (*T_IPC_EVENT_FUNC)
    (T_IPC_EVENT event, T_IPC_EVENT_DATA data, T_PTR arg);

typedef struct T_IPC_EVENT_DATA_STRUCT T_IPC_EVENT_DATA_STRUCT,
    *T_IPC_EVENT_DATA;
struct T_IPC_EVENT_DATA_STRUCT {
    T_IPC_EVENT_TYPE type;
    T_PTR data;
    T_INT4 socket;
    T_IO_CHECK_MODE check_mode;
    T_REAL8 interval;
    T_IPC_CONN conn;
    T_IPC_MSG msg;
};

typedef enum {
    T_IPC_EVENT_USER,
    T_IPC_EVENT_TIMER,
    T_IPC_EVENT_SOCKET,
    T_IPC_EVENT_CONN,
    T_IPC_EVENT_MSG,
    T_IPC_EVENT_MSG_TYPE
} T_IPC_EVENT_TYPE;
```

Properties Table

The entries in the properties table can be traversed using `TipcPropertiesTraverse`. This is how the required traversal function is defined:

```
typedef T_PTR (*T_IPC_PROPERTY_TRAV_FUNC) (T_PTR key, T_PTR value,
    T_PTR arg);
```

This function returns `NULL` to continue the traversal and a non-null value to stop the traversal. All three arguments to the traversal function are of type `T_PTR`. They are usually cast to application-specific data structures. For more information on using Properties tables, see the `TipcProperties*` functions.

Error Handling

SmartSockets provides some simple error handling for its C API functions. The error handling system provides additional information about why a function call failed. This is most often found in functions that return a TRUE or FALSE status. For function calls that support error handling, the global SmartSockets error number is set to a value that provides additional information about why the function failed. This global error number, which is stored in a per-thread value for multithreaded programs, can be retrieved with `TutErrNumGet` and set with `TutErrNumSet`. To simplify logging or printing notification of errors, a string describing the error can be retrieved with `TutErrStrGet` or `TutErrNumToStr`. The Diagnostics section of each function description describes how error handling is supported for that function. Use the error number immediately after a function which supports error handling returns a failure value. At any other time, the value of the error number is meaningless.

For a list of SmartSockets error codes, numbers and descriptions, see the *TIBCO SmartSockets API Quick Reference*.

Chapter 2 **TipcBuf***

This chapter covers the TipcBuf* API functions:

- *TipcBufMsgAppend*
- *TipcBufMsgNext*

TipcBufMsgAppend

Name	TipcBufMsgAppend — append a message to a T_BUF object
Synopsis	<pre>T_BOOL TipcBufMsgAppend(buf, msg) T_BUF buf; T_IPC_MSG msg;</pre>
Arguments	<i>buf</i> — buffer to hold the message <i>msg</i> — msg to append
Return Values	TRUE if the append was successful, FALSE if not.
Diagnostics	If TipcMsgMsgAppend fails, it returns FALSE. It does not set the global error number.
Description	Appends a normal SmartSockets message to a T_BUF object created by TutBufCreate.
Caution	This function is only needed under unusual circumstances, such as for storing a complete SmartSockets message, rather than the contents of the message, in a database.
See Also	TipcBufMsgNext; see the <i>TIBCO SmartSockets Utilities</i> for information on TutBufCreate and TutBufDestroy.
Examples	This example creates a T_BUF object and a message, then stores the message in the buffer: <pre>T_IPC_MSG msg; T_IPC_MT mt; T_BUF buf; buf = TutBufCreate(128); if (buf == NULL) { return; /* error */ } mt = TipcMtLookupByNum(T_MT_INFO); if (mt == NULL) { return; /* error */ } msg = TipcMsgCreate(mt); if (msg == NULL) { return; /* error */ } TipcMsgAppendStr(msg, "hiya!"); TipcBufMsgAppend(buf, msg);</pre>

TipcBufMsgNext

Name	TipcBufMsgNext — get a message from a T_BUF object
Synopsis	<pre>T_IPC_MSG TipcBufMsgNext(buf) T_BUF buf;</pre>
Arguments	<i>buf</i> — buffer holding the message
Return Values	The retrieved message, or NULL if there is an error.
Diagnostics	If TipcBufMsgNext fails, it returns NULL, and sets the global error number to one of: • T_ERR_NULL_PTR — <i>buf</i> was null. • T_ERR_MSG_CORRUPT — the binary large object block cannot be converted into a message. • T_ERR_VAL_TOO_SMALL — the buffer does not contain a complete message.
Description	Retrieves a message from a T_BUF object. The message must have been previously appended to that buffer with TipcBufMsgAppend.
Caution	This function is only necessary under unusual circumstances, such as when storing a complete SmartSockets message in a database, rather than the contents of the message.
See Also	TipcBufMsgAppend; see the <i>TIBCO SmartSockets Utilities</i> for information on TutBufDestroy and TutBufCreate.
Examples	This example pulls a message from a T_BUF object: <pre>T_IPC_MSG msg; msg = TipcBufMsgNext(buf); if (msg == NULL) { return; /* error */ }</pre>

Chapter 3

TipcCbConnProcessGmdFailure - TipcConnWriteCbLookup

This chapter covers the TipcCb and TipcConn API functions:

- *TipcCbConnProcessGmdFailure*
- *TipcCbConnProcessKeepAliveCall*
- *TipcCbSrvError*
- *TipcCbSrvProcessControl*
- *TipcCbSrvProcessGmdFailure*
- *TipcConnAccept*
- *TipcConnBufferGetReadSize*
- *TipcConnBufferGetWriteSize*
- *TipcConnCheck*
- *TipcConnCreate*
- *TipcConnCreateClient*
- *TipcConnCreateServer*
- *TipcConnDefaultCbCreate*
- *TipcConnDefaultCbLookup*
- *TipcConnDestroy*
- *TipcConnErrorCbCreate*
- *TipcConnErrorCbCreate*
- *TipcConnErrorCbLookup*
- *TipcConnFlush*
- *TipcConnGetArch*
- *TipcConnGetAutoFlushSize*
- *TipcConnGetBlockMode*
- *TipcConnGetGmdMaxSize*
- *TipcConnGetGmdNumPending*

- *TipcConnGetNode*
- *TipcConnGetNumQueued*
- *TipcConnGetPid*
- *TipcConnGetSocket*
- *TipcConnGetTimeout*
- *TipcConnGetUniqueSubject*
- *TipcConnGetUser*
- *TipcConnGetXtSource*
- *TipcConnGmdFileCreate*
- *TipcConnGmdFileDelete*
- *TipcConnGmdMsgDelete*
- *TipcConnGmdMsgResend*
- *TipcConnGmdResend*
- *TipcConnKeepAlive*
- *TipcConnLock*
- *TipcConnMainLoop*
- *TipcConnMsgInsert*
- *TipcConnMsgNext*
- *TipcConnMsgProcess*
- *TipcConnMsgSearch*
- *TipcConnMsgSearchType*
- *TipcConnMsgSend*
- *TipcConnMsgSendRpc*
- *TipcConnMsgWrite*
- *TipcConnMsgWriteVa*
- *TipcConnProcessCbCreate*
- *TipcConnProcessCbLookup*
- *TipcConnQueueCbCreate*
- *TipcConnQueueCbLookup*
- *TipcConnRead*

- *TipcConnReadCbCreate*
- *TipcConnReadCbLookup*
- *TipcConnSetAutoFlushSize*
- *TipcConnSetBlockMode*
- *TipcConnSetGmdMaxSize*
- *TipcConnSetSocket*
- *TipcConnSetTimeout*
- *TipcConnTrafficGetBytesRecv8*
- *TipcConnTrafficGetBytesSent8*
- *TipcConnTrafficGetMsgsRecv8*
- *TipcConnTrafficGetMsgsSent8*
- *TipcConnUnlock*
- *TipcConnWriteCbCreate*
- *TipcConnWriteCbLookup*

TipcCbConnProcessGmdFailure

Name	TipcCbConnProcessGmdFailure — callback on all connections to process GMD_FAILURE messages when guaranteed message delivery fails
Synopsis	<pre>void TipcCbConnProcessGmdFailure(conn, data, arg) T_IPC_CONN conn; T_IPC_CONN_PROCESS_CB_DATA data; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection where guaranteed message delivery failed <i>data</i> — callback data <i>arg</i> — callback argument (not used)
Return Values	None
Diagnostics	None
Description	TipcCbConnProcessGmdFailure is a callback that processes each GMD_FAILURE message that is generated internally when guaranteed message delivery fails. When a message with a delivery mode of T_IPC_DELIVERY_SOME or T_IPC_DELIVERY_ALL is sent through a connection (always with TipcConnMsgSend), a copy of the message is saved in the connection’s GMD area. The copy is removed when the sender receives acknowledgment from the receiving processes of successful delivery. If GMD fails, a GMD_FAILURE message is created internally and TipcConnMsgProcess is called to asynchronously notify the sending process that there has been a GMD failure. When a GMD_FAILURE message is processed by a sender, one of two scenarios is possible: • If the sender process does not want to resend the message, it uses TipcConnGmdMsgDelete to terminate GMD for the message • If the sender process does want to resend the message, it takes some user-defined action to correct the problem, then uses TipcConnGmdMsgResend to resend the message Each GMD_FAILURE message contains four fields: • a MSG field containing the message sent by this process where GMD failed • a STR field containing the name of the receiving process where GMD failed (the value of the Unique_Subject option in the receiving process) • an INT4 field containing a SmartSockets error number describing how GMD failed

- a REAL8 field containing the wall time the failed message was originally sent

There are several ways GMD can fail. These error numbers are used only in GMD_FAILURE messages, not in the global SmartSockets global error number accessible with TutErrNumGet:

- `T_ERR_GMD_SENDER_TIMEOUT` — Guaranteed message delivery failed because the message was not acknowledged in time to meet the connection delivery timeout property. This error number is applicable for both `TipcConn* GMD` and `TipcSrv* GMD`. When `TipcConnCheck` and `TipcConnRead` are called, they check how many seconds have elapsed since each guaranteed message was sent. See `TipcConnCheck` and `TipcConnRead` for more information on delivery timeout checking.
- `T_ERR_GMD_RECEIVER_TIMEOUT` — Guaranteed message delivery failed because a receiving `RTclient` (with the option `Server_Disconnect_Mode` set to `warm`) disconnected from `RTserver` but did not reconnect in time to meet the `RTserver` option `Client_Reconnect_Timeout`. This error number is applicable only for `TipcSrv* GMD`.
- `T_ERR_GMD_RECEIVER_EXIT` — Guaranteed message delivery failed because a receiving `RTclient` (with the option `Server_Disconnect_Mode` set to `gmd_failure`) disconnected from `RTserver`. This error number is applicable only for `TipcSrv* GMD`.
- `T_ERR_GMD_NO_RECEIVERS` — Guaranteed message delivery failed because there were no receiving `RTclients` for the message and the `RTserver` option `Zero_Recv_Gmd_Failure` was set to `TRUE`. This error number is applicable only for `TipcSrv* GMD`.

`TipcCbConnProcessGmdFailure` is intended as a sample callback which is designed only to warn the user that guaranteed delivery of a message has failed. More sophisticated applications should create their own process callbacks for GMD_FAILURE messages to perform actions such as user-defined recovery procedures.

`TipcCbConnProcessGmdFailure` first prints output with `TutWarning` that is similar to this output:

```
WARNING: Guaranteed message delivery failed (error number
T_ERR_GMD_ERROR) on the connection <client:local:_node:RTSERVER>.
WARNING: The message was originally sent at TIME.
```

`TipcCbConnProcessGmdFailure` then prints some error-code-specific output with `TutWarning`, prints the undelivered message with `TipcMsgPrint` and `TutOut`, and finally calls `TipcConnGmdMsgDelete` to terminate GMD for the message.

`TipcConnCreate` calls `TipcConnProcessCbCreate` to create a process callback with message type `GMD_FAILURE`, function `TipcCbConnProcessGmdFailure`, and argument `NULL`. This callback may be destroyed by looking up the callback with `TipcConnProcessCbLookup` and then destroying it with `TutCbDestroy`.

`T_ERR_GMD_SENDER_TIMEOUT` is the only error number applicable for GMD that uses only `TipcConn*` functions. For GMD using `RTclient TipcSrv*` functions, the other error numbers also apply, and additional actions are usually needed to process `GMD_FAILURE` messages. `TipcSrvCreate` creates an additional `GMD_FAILURE` process callback using `TipcCbSrvProcessGmdFailure`. See `TipcCbSrvProcessGmdFailure` for more information on the `RTclient`-specific aspects of GMD failure.

Caution `TipcCbConnProcessGmdFailure` is not meant to be called directly and should only be used as a callback function.

See Also `TipcCbSrvProcessGmdFailure`, `TipcConnCreate`, `TipcConnCheck`, `TipcConnProcessCbCreate`, `TipcConnRead`

Examples This example creates a client connection to a server process, looks up the callback that uses `TipcCbConnProcessGmdFailure`, destroys the callback, and creates a new process callback for `GMD_FAILURE` messages:

```
T_IPC_CONN conn;
T_IPC_MT mt;
T_CB cb;
conn = TipcConnCreateClient("local:_node:test.tmp");
if (conn == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_GMD_FAILURE);
if (mt == NULL) {
    return; /* error */
}

cb = TipcConnProcessCbLookup(conn, mt,
                             TipcCbConnProcessGmdFailure, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}

cb = TipcConnProcessCbCreate(conn, mt, my_gmd_failure_cb_func,
                             NULL);
if (cb == NULL) {
    return; /* error */
}
```

TipcCbConnProcessKeepAliveCall

Name	TipcCbConnProcessKeepAliveCall — callback to process KEEP_ALIVE_CALL messages from other end of connection
Synopsis	<pre>void TipcCbConnProcessKeepAliveCall(conn, data, arg) T_IPC_CONN conn; T_IPC_CONN_PROCESS_CB_DATA data; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection that KEEP_ALIVE_CALL message arrived on <i>data</i> — callback data <i>arg</i> — callback argument (printed with TutOut if non-null)
Return Values	None
Diagnostics	None
Description	TipcCbConnProcessKeepAliveCall processes a KEEP_ALIVE_CALL message that is received from the other end of a connection. A keep alive is an explicit attempt to check the health of a connection. See TipcConnKeepAlive for more information on keep alives. TipcConnCreate calls TipcConnProcessCbCreate to create a process callback with message type KEEP_ALIVE_CALL, function TipcCbConnProcessKeepAliveCall, and argument NULL. The argument may be changed by looking up the callback with TipcConnProcessCbLookup and then changing the callback argument with TutCbSetArgument. TipcCbConnProcessKeepAliveCall first examines the argument <i>arg</i>. If <i>arg</i> is not null, it is treated as a string containing a diagnostic output message and is printed by calling this code (which is useful for testing): <pre>TutOut("%s", arg);</pre> TipcCbConnProcessKeepAliveCall then creates a KEEP_ALIVE_RESULT message, sends the message through the connection, and flushes the connection.
Caution	TipcCbConnProcessKeepAliveCall is not meant to be called directly and should only be used as a callback function.
See Also	TipcConnCreate, TipcConnKeepAlive, TipcConnProcessCbCreate

Examples This example creates a client connection to a server, looks up the callback that uses TipcCbConnProcessKeepAliveCall, and changes the callback argument so that TipcCbConnProcessKeepAliveCall prints some output every time a keep alive is received:

```
T_IPC_CONN conn;
T_IPC_MT mt;
T_CB cb;

conn = TipcConnCreateClient("local:_node:test.tmp");
if (conn == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_KEEP_ALIVE_CALL);
if (mt == NULL) {
    return; /* error */
}

cb = TipcConnProcessCbLookup(conn, mt,
                             TipcCbConnProcessKeepAliveCall,
                             NULL);

if (cb == NULL) {
    return; /* error */
}

if (!TutCbSetArgument(cb, "***Received a keep alive***.\n")) {
    return; /* error */
}
```

TipcCbSrvError

Name	TipcCbSrvError — callback to handle errors on the connection to RTserver
Synopsis	<pre>void TipcCbSrvError(conn, data, arg) T_IPC_CONN conn; T_IPC_CONN_ERROR_CB_DATA data; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection to RTserver <i>data</i> — callback data <i>arg</i> — callback argument (not used)
Return Values	None
Diagnostics	None
Description	TipcCbSrvError handles errors on the connection to RTserver. TipcCbSrvError first calls TipcSrvDestroy(T_IPC_SRV_CONN_WARM) to destroy the current connection to RTserver (but leave most information in a warm connection). TipcCbSrvError then calls TipcSrvCreate to create a new connection to RTserver. If the call to TipcSrvCreate fails, the warm connection remains and can be used later to create a new connection. See TipcSrvCreate for more information on warm connections to RTserver. TipcSrvCreate calls TipcSrvErrorCbCreate to create an error callback with function TipcCbSrvError and argument NULL.
Caution	TipcCbSrvError is not meant to be called directly and should only be used as a callback function.
See Also	TipcSrvCreate, TipcSrvDestroy, TipcSrvErrorCbCreate, TipcConnGmdResend
Examples	This example creates a connection to RTserver, changes the socket in the connection to an invalid socket, and attempts to read from the connection. This causes TipcCbSrvError to be called and shows how TipcCbSrvError recreates the connection to RTserver when an error occurs: <pre>if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) { return; /* error */ } /* 200000 is an impossibly large file descriptor */ if (!TipcSrvSetSocket(200000)) { return; /* error */ } TipcSrvRead(0.0);</pre>

The output from the above fragment would be similar to this output:

```
Connecting to project <rtworks> on <_node> RTserver.  
Using local protocol.  
Message from RTserver: Connection established.  
Start subscribing to subject </_workstation1_23506>.  
WARNING: Lost connection to RTserver: error number = 1.  
close: Bad file number  
WARNING: TipcSrvDestroy: could not close socket.  
Attempting to reconnect to RTserver.  
Connecting to project <rtworks> on <_node> RTserver.  
Using local protocol.  
Message from RTserver: Connection established.  
Start subscribing to subject </_workstation1_23506> again.
```


TipcCbSrvProcessControl

Name	TipcCbSrvProcessControl — callback to process CONTROL messages from RTserver
Synopsis	<pre>void TipcCbSrvProcessControl(conn, data, arg) T_IPC_CONN conn; T_IPC_CONN_PROCESS_CB_DATA data; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection to RTserver <i>data</i> — callback data <i>arg</i> — callback argument (not used)
Return Values	None
Diagnostics	If TipcCbSrvProcessControl cannot parse the CONTROL message, it outputs a warning message with TutWarning and then calls TipcMsgPrintError.
Description	TipcCbSrvProcessControl processes a CONTROL message that is received from RTserver. This callback function uses the value of the option Enable_Control_Msgs to check if the command is enabled, and if allowed, it calls TutCommandParseStr with the one string field of the message. TipcSrvCreate calls TipcSrvProcessCbCreate to create a process callback with message type CONTROL, function TipcCbSrvProcessControl, and argument NULL.
Caution	TipcCbSrvProcessControl is not meant to be called directly and should only be used as a callback function.
See Also	TipcSrvCreate, TipcSrvProcessCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCommandParseStr.

Examples This example creates a connection to RTserver, looks up the callback that uses TipcCbSrvProcessControl, and destroys the callback so that CONTROL messages are ignored:

```
T_IPC_MT mt;
T_CB cb;

if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_CONTROL);
if (mt == NULL) {
    return; /* error */
}

cb = TipcSrvProcessCbLookup(mt, TipcCbSrvProcessControl, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}
```

TipcCbSrvProcessGmdFailure

Name	TipcCbSrvProcessGmdFailure — callback on connection to RTserver to process GMD_FAILURE messages when guaranteed message delivery fails
Synopsis	<pre>void TipcCbSrvProcessGmdFailure(conn, data, arg) T_IPC_CONN conn; T_IPC_CONN_PROCESS_CB_DATA data; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection to RTserver <i>data</i> — callback data <i>arg</i> — callback argument (not used)
Return Values	None
Diagnostics	None
Description	TipcCbSrvProcessGmdFailure is a callback that partially processes each GMD_FAILURE message that is generated internally when guaranteed message delivery fails on the connection to RTserver. (TipcCbConnProcessGmdFailure does the rest of the processing.) TipcCbSrvProcessGmdFailure works in conjunction with TipcCbConnProcessGmdFailure to perform all standard processing of GMD_FAILURE messages. TipcCbSrvProcessGmdFailure calls TipcSrvGmdMsgServerDelete to inform RTserver to terminate GMD for the message. See TipcCbConnProcessGmdFailure for more information on processing GMD_FAILURE messages. TipcCbSrvProcessGmdFailure is a sample callback designed only to inform RTserver to terminate GMD for the message. More sophisticated applications should create their own process callbacks for GMD_FAILURE messages to perform actions such as user-defined recovery procedures. TipcSrvCreate calls TipcSrvProcessCbCreate to create a process callback with message type GMD_FAILURE, function TipcCbSrvProcessGmdFailure, and argument NULL. This callback may be destroyed by looking up the callback with TipcSrvProcessCbLookup and then destroying it with TutCbDestroy.
Caution	TipcCbSrvProcessGmdFailure is not meant to be called directly and should only be used as a callback function.
See Also	TipcCbConnProcessGmdFailure, TipcSrvGmdMsgServerDelete

Examples This example connects to RTserver, looks up the callback that uses TipcCbSrvProcessGmdFailure, destroys the callback, and creates a new process callback for GMD_FAILURE messages:

```
T_IPC_MT mt;
T_CB cb;

if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_GMD_FAILURE);
if (mt == NULL) {
    return; /* error */
}

cb = TipcSrvProcessCbLookup(mt, TipcCbSrvProcessGmdFailure, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}

cb = TipcSrvProcessCbCreate(mt, my_gmd_failure_cb_func, NULL);
if (cb == NULL) {
    return; /* error */
}
```

TipcConnAccept

Name	TipcConnAccept — accept connection from client
Synopsis	<pre>T_IPC_CONN TipcConnAccept(server_conn) T_IPC_CONN server_conn;</pre>
Arguments	<i>server_conn</i> — server connection
Return Values	Connection to new client if successful, NULL otherwise.
Diagnostics	If TipcConnAccept fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>server_conn</i> was null • T_ERR_CONN_INIT_FAILED — initialization with the other end of the connection failed (the two ends of the connection are not compatible) • any error number from TutSocketAccept
Description	TipcConnAccept accepts a connection from a new client by calling TutSocketAccept and returns a new connection to that client. The original connection is left open to accept more clients. TipcConnAccept blocks waiting for a client to connect if the block mode of the connection is TRUE. See TipcConnGetBlockMode for information on connection block modes. TipcConnAccept also sets the size of the new connection's socket send and receive buffers to 48K bytes with the function setsockopt.
Caution	To avoid blocking if no client is waiting to connect, use TipcConnCheck before calling TipcConnAccept.
See Also	TipcConnCheck; see the <i>TIBCO SmartSockets Utilities</i> for information on TutSocketAccept.
Examples	This example uses TipcConnCheck to wait for up to 10 seconds for a new client to connect, then accepts the new client with TipcConnAccept: <pre>if (TipcConnCheck(server_conn, T_IO_CHECK_READ, 10.0)) { client_conn = TipcConnAccept(server_conn); if (client_conn == NULL) { return; /* error */ } } else { return; /* error */ }</pre>

TipcConnBufferGetReadSize

Name TipcConnBufferGetReadSize — get the total number of bytes in a connection’s read buffer

Synopsis `T_BOOL TipcConnBufferGetReadSize(conn, read_size_return)`
 `T_IPC_CONN conn;`
 `T_INT4 *read_size_return;`

Arguments *conn* — connection to get information for
 read_size_return — returned number of bytes

Return Values TRUE if the number of bytes was sucessfully retrieved, FALSE otherwise.

Diagnostics If TipcConnBufferGetReadSize fails, it returns FALSE and sets the global SmartSockets error number to:

- `T_ERR_NULL_PTR` — *conn* or *read_size_return* was null

Description TipcConnBufferGetReadSize retrieves the total number of bytes in a connection’s read buffer.

See Also TipcConnBufferGetWriteSize

Examples This example retrieves the number of bytes in a connection’s read buffer:

```
T_IPC conn;
T_INT4 read_size;
.
.
.
if (!TipcConnBufferGetReadSize(conn, &read_size)) {
    return; /* error */
}
```

TipcConnBufferGetWriteSize

Name	TipcConnBufferGetWriteSize — get the total number of bytes in a connection's write buffer
Synopsis	<pre>T_BOOL TipcConnBufferGetWriteSize(conn, write_size_return) T_IPC_CONN conn; T_INT4 *write_size_return;</pre>
Arguments	<i>conn</i> — connection to get information for <i>write_size_return</i> — returned number of bytes
Return Values	TRUE if the number of bytes was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcConnBufferGetWriteSize fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>conn</i> or <i>write_size_return</i> was null
Description	TipcConnBufferGetWriteSize retrieves the total number of bytes in a connection's write buffer. The returned number is usually the same as the total number of bytes actually sent on the connection, but sometimes the returned number is larger than the number of bytes sent when the write buffer is not frequently flushed.
See Also	TipcConnBufferGetReadSize
Examples	This example retrieves the number of bytes in a connection's write buffer: <pre>T_IPC conn; T_INT4 write_size; . . . if (!TipcConnBufferGetWriteSize(conn, &write_size)) { return; /* error */ }</pre>

TipcConnCheck

Name	TipcConnCheck — check if data can be read from or written to a connection
Synopsis	<pre>T_BOOL TipcConnCheck(conn, check_mode, timeout) T_IPC_CONN conn; T_IO_CHECK_MODE check_mode; T_REAL8 timeout;</pre>
Arguments	<i>conn</i> — connection to check <i>check_mode</i> — how to check the connection for availability (T_IO_CHECK_READ to check for reading, T_IO_CHECK_WRITE to check for writing) <i>timeout</i> — maximum number of seconds to wait for availability
Return Values	TRUE if data can be read or written, FALSE if timeout reached or error occurred.
Diagnostics	If TipcConnCheck fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR— <i>conn</i> was null • T_ERR_VAL_INVALID — <i>check_mode</i> was not a valid T_IO_CHECK_MODE • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_TIMEOUT_REACHED — the <i>timeout</i> period was reached • T_ERR_FAILURE_DETECTED — a network failure was detected • any error number from TipcConnFlush • any error number from TutSocketCheck
Description	TipcConnCheck checks if data can be read from or written to a connection by calling TutSocketCheck with the connection's socket. Use 0.0 for timeout to poll the connection and return immediately. Use T_TIMEOUT_FOREVER for timeout to wait indefinitely for availability. If <i>check_mode</i> is T_IO_CHECK_READ the connection is checked for reading. In this mode, if the connection's auto flush size is not T_IPC_NO_AUTO_FLUSH, then TipcConnCheck first calls TipcConnFlush to flush the connection's write buffer (so that any responses to the outgoing data are available sooner). If the connection's block mode is TRUE or the connection's read timeout is 0.0, then TipcConnCheck calls TutSocketCheck once with timeout and returns. Otherwise

TipcConnCheck keeps calling TutSocketCheck over and over with a calculated timeout until data is available, the timeout period is reached, or an error occurs. The calculated timeout ensures that TipcConnCheck never waits for more than *read_timeout* seconds past the last time data was read on the connection.

This pseudocode shows the calculation:

```
$elapsed_time = TutGetWallTime() - (the time when data was last
read)
if ($elapsed_time > read_timeout) {
    $calculated_timeout = 0
}
else {
    $calculated_timeout = read_timeout - $elapsed_time
}
```

After each call to TutSocketCheck with the calculated timeout, if no data is available, and *read_timeout* seconds have elapsed since the last time data was read, then a keep alive is initiated to check the health of the connection by calling TipcConnKeepAlive. If TipcConnKeepAlive fails, a network failure has occurred, and the connection's error callbacks are called.

If no data is available for reading, TipcConnCheck checks for guaranteed message delivery timeouts. When messages with a delivery mode of `T_IPC_DELIVERY_SOME` or `T_IPC_DELIVERY_ALL` are sent through a connection with TipcConnMsgSend, copies of the messages are saved in the connection GMD area. The copies are removed when acknowledgment of delivery is received by the sender from the receiving processes. TipcConnCheck checks how many seconds have elapsed since each previously sent guaranteed message was sent. For each message where the elapsed time is greater than the connection delivery timeout, the timed-out message is removed from the connection GMD area, a `GMD_FAILURE` message is created whose first field is the timed-out message, and TipcConnMsgProcess is called to asynchronously notify the sending process that there has been a delivery timeout. See TipcCbConnProcessGmdFailure for more information on processing `GMD_FAILURE` messages. Because TipcConnCheck only checks for delivery timeouts after calling TutSocketCheck, ensure that the timeout used with TipcConnCheck is not larger than the connection delivery timeout property.

If *check_mode* is `T_IO_CHECK_WRITE`, the connection is checked for writing. If the connection's block mode is `TRUE` or the connection's write timeout is `0.0`, then TipcConnCheck calls TutSocketCheck once with timeout and returns. Otherwise TipcConnCheck calls TutSocketCheck once with a calculated timeout that is similar to the one used when checking for reading (see above pseudocode). Note that checking for writing does not initiate keep alives; if data cannot be written in time, there is no point in trying a keep alive (which would send a `KEEP_ALIVE_CALL` message). If the check for writing fails, a network failure has occurred, and the connection's error callbacks are called.

Caution TipcConnCheck is a low-level function that is normally not used directly by developers. TipcConnCheck is normally called only by TipcConnFlush and TipcConnRead.

See Also TipcConnFlush, TipcConnRead; see the *TIBCO SmartSockets Utilities* for information on TutSocketCheck.

Examples This example checks if data is available for reading on a connection within 3.4 seconds:

```
if (!TipcConnCheck(conn, T_IO_CHECK_READ, 3.4)) {  
    return; /* error */  
}
```

TipcConnCreate

Name	TipcConnCreate — create a new connection
Synopsis	T_IPC_CONN TipcConnCreate()
Arguments	None
Return Values	New connection
Diagnostics	None
Description	TipcConnCreate creates a new connection. The connection has a socket file descriptor of -1, a block mode of TRUE, a read buffer of 16384 bytes, a write buffer of 16384 bytes, and an auto flush size of 8192 bytes. TipcConnCreate calls TipcConnProcessCbCreate to create a process callback with message type KEEP_ALIVE_CALL, function TipcCbConnProcessKeepAliveCall, and argument NULL. TipcConnCreate calls TipcConnProcessCbCreate to create a process callback with message type GMD_FAILURE, function TipcCbConnProcessGmdFailure, and argument NULL. TipcConnCreate is called by the functions TipcConnCreateServer and TipcConnCreateClient. TipcConnCreate is also useful for creating a dummy connection, then creating process callbacks, then processing messages.
Caution	None
See Also	TipcConnCreateClient, TipcConnCreateServer

Examples This example creates a connection, creates some process and default callbacks, then reads and processes messages from the message file `data.msg`:

```
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_MSG_FILE msg_file;
T_IPC_MSG msg;

conn = TipcConnCreate();

/* create a process callback for TIME messages */
mt = TipcMtLookupByNum(T_MT_TIME);
if (mt == NULL) {
    return; /* error */
}

if (TipcConnProcessCbCreate(conn, mt, process_time, NULL) == NULL) {
    return; /* error */
}

/* create a process callback for NUMERIC_DATA messages */
mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (TipcConnProcessCbCreate(conn, mt, process_nd, NULL) == NULL) {
    return; /* error */
}

/* create a default callback for all other messages */
if (TipcConnDefaultCbCreate(conn, process_default, NULL) == NULL) {
    return; /* error */
}

/* create a message file that is open for reading */
msg_file = TipcMsgFileCreate("data.msg",
T_IPC_MSG_FILE_CREATE_READ);
if (msg_file == NULL) {
    return; /* error */
}

/* read and process messages */
while (TipcMsgFileRead(msg_file, &msg)) {
    if (!TipcConnMsgProcess(conn, msg)) {
        return; /* error */
    }
    if (!TipcMsgDestroy(msg)) {
        return; /* error */
    }
}
```

TipcConnCreateClient

Name	TipcConnCreateClient — create the client side of a connection
Synopsis	<pre>T_IPC_CONN TipcConnCreateClient(logical_conn_name) T_STR logical_conn_name;</pre>
Arguments	<i>logical_conn_name</i> — logical connection name that the server process is using
Return Values	New connection if successful, NULL otherwise.
Diagnostics	If TipcConnCreateClient fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>logical_conn_name</i> was null • T_ERR_CONN_INIT_FAILED — initialization with the other end of the connection failed (the two ends of the connection are not compatible) • any error number from TutSocketCreateClient*
Description	TipcConnCreateClient creates a client connection and connects to a server connection (which must have already been created). When creating a client/server connection, the server side must always be created first. The client side must have something to connect to. SmartSockets simplifies the creation of connections with logical connection names that are specified consistently for all protocols. A server process uses a logical connection name to create a server connection, and a client process uses the same logical connection name to create a client connection to the server process. Each logical connection name has the form <code>protocol:node:address</code>. For the client process to find the server process, the logical connection name used by the client must exactly match the logical connection name used by the server. (For example, the name <code>tcp:workstation1:1234</code> does not match the name <code>tcp:workstation2:1234</code>.) See the <i>TIBCO SmartSockets User's Guide</i> for more details on logical connection names. TipcConnCreateClient also sets the size of the new connection's socket send and receive buffers to 48K bytes with the function <code>setsockopt</code>.
Caution	None
See Also	TipcConnCreateServer. See the <i>TIBCO SmartSockets Utilities</i> for information on TutSocketCreateClient*.

Examples This example connects to a server connection on the same node and sends a NUMERIC_DATA message to the server:

```

/* create connection to server */
conn = TipcConnCreateClient("tcp:_node:4242");
if (conn == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

/* now we can send/rcv messages on connection */
if (!TipcConnMsgWrite(conn, mt,
                      T_IPC_FT_STR, "speed_limit",
                      T_IPC_FT_REAL8, 55.0,
                      NULL)) {
    return; /* error */
}

```

TipcConnCreateServer

Name	TipcConnCreateServer — create the server side of a connection
Synopsis	<pre>T_IPC_CONN TipcConnCreateServer(logical_conn_name) T_STR logical_conn_name;</pre>
Arguments	<i>logical_conn_name</i> — logical connection name to associate with this connection
Return Values	New connection if successful, NULL otherwise.
Diagnostics	If TipcConnCreateServer fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>logical_conn_name</i> was null • any error number from TutSocketCreateServer*
Description	TipcConnCreateServer creates a server connection that can accept new clients by calling TipcConnAccept. When creating a client/server connection, the server side must always be created first (the client side must have something to connect to). After creating a server connection, TipcConnAccept can be used to accept a new client, and TipcConnCheck can be used to check if a new client is available. SmartSockets simplifies the creation of connections with logical connection names that are specified consistently for all protocols. A server process uses a logical connection name to create a server connection, and a client process uses the same logical connection name to create a client connection to the server process. Each logical connection name has the form protocol:node:address. For the client process to find the server process, the logical connection name used by the client must exactly match the logical connection name used by the server. (For example, the name tcp:workstation1:1234 does not match the name tcp:workstation2:1234.) See the <i>TIBCO SmartSockets User's Guide</i> for more details on logical connection names.
Caution	None
See Also	TipcConnAccept, TipcConnCreateClient; see the <i>TIBCO SmartSockets Utilities</i> for information on TutSocketCreateServer*.

Examples This example creates a server connection, accepts a new client, and sends a NUMERIC_DATA message to the client:

```

/* create connection for clients to connect on */
server_conn = TipcConnCreateServer("tcp:_node:4242");
if (server_conn == NULL) {
    return; /* error */
}

/* accept connection from client */
accept_conn = TipcConnAccept(server_conn);
if (accept_conn == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

/* now we can send/recv messages on accept_conn */
if (!TipcConnMsgWrite(accept_conn, mt,
                     T_IPC_FT_STR, "speed_limit",
                     T_IPC_FT_REAL8, 55.0,
                     NULL)) {
    return; /* error */
}

```


TipcConnDefaultCbCreate

Name	TipcConnDefaultCbCreate — create a default callback in a connection
Synopsis	<pre>T_CB TipcConnDefaultCbCreate(<i>conn</i>, <i>func</i>, <i>arg</i>) T_IPC_CONN <i>conn</i>; T_IPC_CONN_DEFAULT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>conn</i> — connection for which to create callback <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcConnDefaultCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcConnDefaultCbCreate creates a default callback in a connection. These callbacks are called by TipcConnMsgProcess when there are no process callbacks for the type of message being processed. Default callbacks are useful for processing unexpected message types or for generic processing of most message types. See TipcConnMsgProcess for more information on the relationship between process callbacks and default callbacks.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnMsgProcess, TipcConnDefaultCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example creates a default callback which warns that an unexpected message has arrived:

```
void T_ENTRY my_conn_default_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_DEFAULT_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    if (!TipcMsgPrintError(data->msg)) {
        return; /* error */
    }
} /* my_conn_default_cb */

/* ===== */
/*...code from calling function is below */

/* create default callback */
if (TipcConnDefaultCbCreate(conn, my_conn_default_cb, NULL) ==
NULL) {
    return; /* error */
}
```

TipcConnDefaultCbLookup

Name	TipcConnDefaultCbLookup — look up a default callback in a connection
Synopsis	<pre>T_CB TipcConnDefaultCbLookup(<i>conn</i>, <i>func</i>, <i>arg</i>) T_IPC_CONN <i>conn</i>; T_IPC_CONN_DEFAULT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>conn</i> — connection to look up callback for <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcConnDefaultCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcConnDefaultCbLookup looks up a default callback with the desired function and argument in a connection. These callbacks are called by TipcConnMsgProcess when there are no process callbacks for the type of message being processed. See TipcConnDefaultCbCreate for more information on default callbacks.
Caution	None
See Also	TipcConnDefaultCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up and destroys a default callback: <pre>cb = TipcConnDefaultCbLookup(conn, my_conn_default_cb, NULL); if (cb == NULL) { return; /* error */ } if (!TutCbDestroy(cb)) { return; /* error */ }</pre>

TipcConnDestroy

Name	TipcConnDestroy — destroy a connection
Synopsis	<pre>T_BOOL TipcConnDestroy(conn) T_IPC_CONN conn;</pre>
Arguments	<i>conn</i> — connection to destroy
Return Values	TRUE if connection was successfully destroyed, FALSE otherwise.
Diagnostics	If TipcConnDestroy fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>conn</i> was null
Description	TipcConnDestroy destroys a connection. All memory for the connection is freed, and the connection’s socket is closed. All guaranteed message delivery files are closed, but the files on disk are not removed.
Caution	TipcConnDestroy does not try to flush the connection’s write buffer. Use TipcConnFlush before calling TipcConnDestroy to get this behavior. TipcConnDestroy does not try to acknowledge any unacknowledged messages with a delivery mode of T_IPC_DELIVERY_SOME or T_IPC_DELIVERY_ALL that have been read from the connection. Use TipcMsgAck or TipcMsgDestroy before calling TipcConnDestroy to get this behavior. Use TipcConnDestroy with caution from a connection callback such as a connection process callback. Because TipcConnDestroy completely destroys the connection, the program crashes if control from the callback is returned to a function such as TipcConnMainLoop. A process with connections using the tcp protocol may lose outgoing messages if the process terminates without calling TipcConnDestroy to destroy each connection. TCP / IP’s SO_LINGER option, which preserves data, is ignored when closing a socket that has non-blocking I / O enabled. While data loss rarely occurs on UNIX and OpenVMS, it can happen very frequently on Windows. TipcConnDestroy sets the block mode of the connection to FALSE before closing the connection’s socket, which forces the operating system to deliver all flushed outgoing messages.
See Also	TipcConnCreate

Examples TipcConnDestroy is useful when you use TipcConnAccept to accept a connection from a client, and then need to destroy the original connection (so that no more clients can connect):

```

/* create server side */
server_conn = TipcConnCreateServer("tcp:_node:4242");
if (server_conn == NULL) {
    return; /* error */
}

/* wait for client */
client_conn = TipcConnAccept(server_conn);
if (client_conn == NULL) {
    return; /* error */
}

/* destroy server connection */
if (!TipcConnDestroy(server_conn)) {
    return; /* error */
}

```

TipcConnErrorCbCreate

Name	TipcConnErrorCbCreate — create an error callback in a connection
Synopsis	<pre>T_CB_TipcConnErrorCbCreate(conn, func, arg) T_IPC_CONN conn; T_IPC_CONN_ERROR_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection for which to create callback <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcConnErrorCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcConnErrorCbCreate creates an error callback in a connection. These callbacks are called when an unrecoverable error occurs on a connection. The possible errors are: • the other end of the connection’s socket is closed, which may occur in TipcConnRead (when the socket function <i>recv</i> returns 0) or TipcConnFlush (when the socket function <i>send</i> returns -1) • the connection read timeout has elapsed without any data being read from the connection’s socket, and then the keep alive failed • the connection write timeout has elapsed without any data being able to be written to the connection’s socket Connection error callbacks usually take measures to re-establish the connection. See <i>TIBCO SmartSockets Utilities</i> for more information on callbacks.
Caution	A connection error callback function should not destroy the connection with TipcConnDestroy. The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnErrorCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example creates a connection error callback which prints a warning and sets a global flag to indicate that an error has occurred:

```
T_BOOL my_conn_error_happened = FALSE;

/* ===== */
/*..my_conn_error_cb -- connection error callback */
void T_ENTRY my_conn_error_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_ERROR_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    TutWarning("Error <%s> occurred on my connection.\n",
               TutErrNumToStr(data->err_num));
    my_conn_error_happened = TRUE;
} /* my_conn_error_cb */

/* ===== */
/*...code from calling fragment is below */

if (TipcConnErrorCbCreate(conn, my_conn_error_cb, NULL) == NULL) {
    return; /* error */
}

/* Now use the global flag to check for errors. */
msg = TipcConnMsgNext(conn, T_TIMEOUT_FOREVER);
if (msg == NULL && my_conn_error_happened) {
    return; /* error */
}
```

TipcConnErrorCbLookup

Name	TipcConnErrorCbLookup — look up an error callback in a connection
Synopsis	<pre>T_CB TipcConnErrorCbLookup(conn, func, arg) T_IPC_CONN conn; T_IPC_CONN_ERROR_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection to look up callback for <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcConnErrorCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcConnErrorCbLookup looks up a connection error callback with the desired function and argument in a connection. These callbacks are called when an unrecoverable error occurs on a connection. See TipcConnErrorCbCreate for more information on connection error callbacks.
Caution	None
See Also	TipcConnErrorCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up a connection error callback and destroys it: <pre>cb = TipcConnErrorCbLookup(conn, my_conn_error_cb, NULL); if (cb == NULL) { return; /* error */ } if (!TutCbDestroy(cb)) { return; /* error */ }</pre>

TipcConnFlush

Name	TipcConnFlush — flush buffered outgoing messages to connection's socket
Synopsis	<pre>T_BOOL TipcConnFlush(conn) T_IPC_CONN conn;</pre>
Arguments	<i>conn</i> — connection to flush
Return Values	TRUE if data was successfully flushed to connection, FALSE otherwise.
Diagnostics	If TipcConnFlush fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null • T_ERR_EOF — the other end of the connection's socket has been closed (EOF has been reached). Destroy the connection and create a new one if necessary. • T_ERR_C — a C error has occurred. Call the function TutErrNumGetC to obtain the related C-specific error value. • T_ERR_OS — an operating system error has occurred. Call the function TutErrNumGetOs to obtain the related OS-specific error value. • T_ERR_SOCKET — a socket error has occurred. Call the function TutErrNumGetSocket to obtain the related socket-specific error value. • T_ERR_WOULD_BLOCK — the connection's block mode is FALSE, and not all data could be sent without blocking • any error number from TipcConnCheck
Description	TipcConnFlush writes all buffered outgoing messages to a connection's socket. Use TipcConnFlush sparingly. Calling it frequently will decrease response time, at the cost of increasing CPU consumption (due to the increased number of calls to the socket send function). If the connection's block mode is TRUE or the connection's write timeout is 0.0, then TipcConnFlush calls send once and returns. Otherwise TipcConnFlush calls send over and over until all data has been written, an error occurs, or the connection write timeout is reached. Examples of such errors include the socket being closed by the process at the other end of the connection. If the connection's auto flush size is not T_IPC_NO_AUTO_FLUSH, it is usually not necessary to call TipcConnFlush. In this case, TipcConnRead and TipcConnCheck always call TipcConnFlush to flush the connection's write buffer. In addition, TipcConnMsgSend automatically calls TipcConnFlush if the number of buffered bytes of data is greater than the connection's auto flush size.

If the connection's auto flush size is 0, it is never necessary to call `TipcConnFlush`. In this case, `TipcConnMsgSend` always calls `TipcConnFlush` to flush the connection's write buffer.

`TipcConnFlush` blocks if `send` blocks (that is, the socket's internal buffer is full). If the flush fails (because `send` returns 0 or -1), the connection's error callbacks are called to clean up, and `TipcConnFlush` returns `FALSE`.

Caution None

See Also `TipcConnCheck`, `TipcConnGetAutoFlushSize`, `TipcConnRead`

Examples This example shows how to use `TipcConnFlush` to flush a message to a connection's socket:

```
if (!TipcConnMsgSend(conn, msg)) {  
    return; /* error */  
}  
  
if (!TipcConnFlush(conn)) {  
    return; /* error */  
}
```

TipcConnGetArch

Name	TipcConnGetArch — get the architecture name of a connection’s peer process
Synopsis	<pre>T_BOOL TipcConnGetArch(conn, arch_return) T_IPC_CONN conn; T_STR *arch_return;</pre>
Arguments	<i>conn</i> — connection to get architecture name from <i>arch_return</i> — storage for connection architecture name
Return Values	TRUE if the architecture name was successfully retrieved from the connection, FALSE otherwise.
Diagnostics	If TipcConnGetArch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>arch_return</i> was null
Description	TipcConnGetArch gets the architecture name of a connection’s peer process (the process that this process is communicating with). The peer process architecture name is useful for diagnostic purposes. The architecture name is the SmartSockets architecture of the form <i>Machine_OperatingSystem</i> (for example, sun4_solaris). This value is stored in: UNIX: the environment variable RTARCH OpenVMS: the logical RTARCH Windows: the environment variable RTARCH
Caution	If TipcConnGetArch returns FALSE, it does not store a value in <i>arch_return</i>. The value stored in <i>arch_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcConnGetNode, TipcConnGetPid, TipcConnGetUniqueSubject, TipcConnGetUser

Examples This code fragment prints the architecture name of a connection's peer process:

```
T_STR arch;

if (!TipcConnGetArch(conn, &arch)) {
    return; /* error */
}

TutOut("Connection peer process architecture name is <%s>.\n",
arch);
```

TipcConnGetAutoFlushSize

Name	TipcConnGetAutoFlushSize — get the auto flush size of a connection
Synopsis	<pre>T_BOOL TipcConnGetAutoFlushSize(conn, auto_flush_size_return) T_IPC_CONN conn; T_INT4 *auto_flush_size_return;</pre>
Arguments	<i>conn</i> — connection to get auto flush size from <i>auto_flush_size_return</i> — storage for auto flush size
Return Values	TRUE if the auto flush size was successfully retrieved from connection, FALSE otherwise.
Diagnostics	If TipcConnGetAutoFlushSize fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>conn</i> was null or <i>auto_flush_size_return</i> was null
Description	TipcConnGetAutoFlushSize gets the auto flush size of a connection. The auto flush size is used to control how much outgoing data (in bytes) is buffered before being automatically flushed to the other end of the connection. TipcConnMsgSend uses the auto flush size to determine when to flush the buffered outgoing messages. If the number of bytes of pending outgoing data exceeds the auto flush size, TipcConnMsgSend calls TipcConnFlush. The default for the auto flush size is 8192 bytes. Using auto flush size 0 disables all buffering and forces each outgoing message to be flushed immediately. Using auto flush size T_IPC_NO_AUTO_FLUSH_SIZE disables all automatic flushing of outgoing messages.
Caution	If TipcConnGetAutoFlushSize returns FALSE, it does not store a value in <i>auto_flush_size_return</i> .
See Also	TipcConnSetAutoFlushSize
Examples	This example gets the auto flush size of a connection with TipcConnGetAutoFlushSize and prints it: <pre>T_INT4 auto_flush_size; if (!TipcConnGetAutoFlushSize(conn, &auto_flush_size)) { return; /* error */ } TutOut("The auto buffer size of the connection is %d bytes.\n", auto_flush_size);</pre>

TipcConnGetBlockMode

Name	TipcConnGetBlockMode — get the block mode of a connection
Synopsis	<pre>T_BOOL TipcConnGetBlockMode(conn, block_mode_return) T_IPC_CONN conn; T_BOOL *block_mode_return;</pre>
Arguments	<i>conn</i> — connection to get block mode for <i>block_mode_return</i> — storage for block mode
Return Values	TRUE if block mode was successfully retrieved from connection, FALSE otherwise.
Diagnostics	If TipcConnGetBlockMode fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>conn</i> was null or <i>block_mode_return</i> was null
Description	TipcConnGetBlockMode gets the block mode of a connection. The block mode controls how TipcConnFlush behaves when a connection’s socket (write) buffer is full, how TipcConnRead behaves when a connection’s socket (read) buffer is empty, and how TipcConnAccept behaves when no new client is waiting to connect. The block mode setting is also related to the connection timeout properties. See TipcConnGetTimeout for more information on these timeout properties. The default block mode is TRUE. In this mode, TipcConnFlush does not return until all data has been written to the connection’s socket (or an error occurs), and TipcConnRead does not return until some data has been read from the connection’s socket, or an error occurs, or a specified amount of time has elapsed. If block mode is FALSE, the connection's socket flags are modified with the C function <code>fcntl</code> to enable non-blocking socket operations. If the connection timeout properties are 0.0, then TipcConnFlush returns FALSE (and sets the global SmartSockets error number to T_ERR_WOULD_BLOCK) if not all buffered data can be sent, and TipcConnRead returns FALSE (and sets the global SmartSockets error number to T_ERR_WOULD_BLOCK) if no data can be read. If the connection timeout properties are greater than 0.0, then the amount of time TipcConnRead and TipcConnFlush can block is limited by the value of the connection properties.
Caution	If TipcConnGetBlockMode returns FALSE, it does not store a value in <i>block_mode_return</i> .
See Also	TipcConnFlush, TipcConnRead, TipcConnSetBlockMode

Examples This example gets the block mode of a connection and prints it:

```
T_BOOL block_mode;

if (!TipcConnGetBlockMode(conn, &block_mode)) {
    return; /* error */
}

TutOut("The connection block mode is %d.\n", block_mode);
```

TipcConnGetGmdMaxSize

Name TipcConnGetGmdMaxSize — get the maximum size of a connection’s GMD area

Synopsis `T_BOOL TipcConnGetGmdMaxSize(conn, gmd_max_size_return)`
`T_IPC_CONN conn;`
`T_UINT4 *gmd_max_size_return;`

Arguments *conn* — connection to get GMD area maximum size from
gmd_max_size_return — storage for connection GMD area maximum size

Return Values TRUE if the GMD area maximum size was successfully retrieved from the connection, FALSE otherwise.

Diagnostics If TipcConnGetGmdMaxSize fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *conn* was null or *gmd_max_size_return* was null
- T_ERR_VAL_INVALID — the connection is not set up for GMD.
TipcConnGmdFileCreate must be called before TipcConnGetGmdMaxSize.

Description TipcConnGetGmdMaxSize gets the GMD area maximum size (in bytes) of a connection. A GMD area holds guaranteed message delivery information for both incoming and outgoing messages. There are two types of guaranteed message delivery: file-based GMD and memory-only GMD. See TipcConnGmdFileCreate for more information on GMD areas.

For file-based GMD, this limit is the maximum file size, and for memory-only GMD this limit is the maximum amount of memory allowed. The default GMD area maximum size is 0, which means that no GMD area maximum size limit checking is performed. The connection GMD area maximum size and the connection delivery timeout can be used to constrain the amount of system resources used for GMD by space and by time.

If the connection GMD area maximum size is exceeded, then no further messages can be sent with guaranteed delivery (TipcConnMsgSend returns FALSE and sets the global SmartSockets error number to T_ERR_VAL_TOO_LARGE) until some unacknowledged previously sent messages are acknowledged.

Caution If TipcConnGetGmdMaxSize returns FALSE, it does not store a value in *gmd_max_size_return*.

See Also TipcConnGmdFileCreate, TipcConnSetGmdMaxSize, TipcConnGetGmdMaxSize

Examples This example prints the GMD area maximum size of a connection:

```
T_UINT4 gmd_max_size;

if (!TipcConnGetGmdMaxSize(conn, &gmd_max_size)) {
    return; /* error */
}

TutOut("Connection GMD max size is %d.\n", gmd_max_size);
```

TipcConnGetGmdNumPending

Name	TipcConnGetGmdNumPending — get the number of outgoing GMD messages still pending on a connection
Synopsis	<pre>T_BOOL TipcConnGetGmdNumPending(conn, gmd_num_pending_return) T_IPC_CONN conn; T_INT4 *gmd_num_pending_return;</pre>
Arguments	<i>conn</i> — connection to get pending message count from <i>gmd_num_pending_return</i> — storage for pending message count
Return Values	TRUE if the number of pending GMD messages was successfully retrieved from the connection, FALSE otherwise.
Diagnostics	If TipcConnGetGmdNumPending fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>conn</i> was null or <i>gmd_num_pending_return</i> was null
Description	TipcConnGetGmdNumPending gets the number of GMD messages sent on the connection that have been neither acknowledged nor explicitly deleted from the GMD area. Messages are deleted from the GMD area when a GMD_ACK message is received and also, by default, when a GMD_FAILURE message is processed.
Caution	If TipcConnGetGmdNumPending returns FALSE, it does not store a value in <i>gmd_num_pending_return</i> .
See Also	TipcSrvGetGmdNumPending
Examples	This example loops processing messages until there are no more outgoing GMD messages pending on a connection: <pre>T_INT4 num_pending; do { if (!TipcConnMainLoop(conn, 1.0)) { return; /* error */ } if (!TipcConnGetGmdNumPending(conn, &num_pending)) { return; /* error */ } } while (num_pending > 0);</pre>

TipcConnGetNode

Name	TipcConnGetNode — get the node name of a connection's peer process
Synopsis	<pre>T_BOOL TipcConnGetNode(conn, node_return) T_IPC_CONN conn; T_STR *node_return;</pre>
Arguments	<i>conn</i> — connection to get node name from <i>node_return</i> — storage for connection node name
Return Values	TRUE if the node name was successfully retrieved from the connection, FALSE otherwise.
Diagnostics	If TipcConnGetNode fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>conn</i> was null or <i>node_return</i> was null
Description	TipcConnGetNode gets the node name of a connection's peer process (the process with which this process is communicating through the connection). The peer process node name can be useful for diagnostic purposes.
Caution	If TipcConnGetNode returns FALSE, it does not store a value in <i>node_return</i>. The value stored in <i>node_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcConnGetArch, TipcConnGetPid, TipcConnGetUniqueSubject, TipcConnGetUser
Examples	This example prints the node name of a connection's peer process: <pre>T_STR node; if (!TipcConnGetNode(conn, &node)) { return; /* error */ } TutOut("Connection peer process node name is <%s>.\n", node);</pre>

TipcConnGetNumQueued

Name	TipcConnGetNumQueued — get number of queued messages from a connection
Synopsis	<pre>T_BOOL TipcConnGetNumQueued(conn, num_queued_return) T_IPC_CONN conn; T_INT4 *num_queued_return;</pre>
Arguments	<i>conn</i> — connection from which to get number of queued messages <i>num_queued_return</i> — storage for number of queued messages
Return Values	TRUE if number of queued messages was successfully retrieved from connection, FALSE otherwise.
Diagnostics	If TipcConnGetNumQueued fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>num_queued_return</i> was null
Description	TipcConnGetNumQueued gets the number of queued messages from a connection. A connection has a priority queue of incoming messages. Messages are inserted into the queue with TipcConnMsgInsert (TipcConnRead uses TipcConnMsgInsert to insert newly read messages), may be removed from the front of the queue with TipcConnMsgNext, and may be removed from anywhere in the queue with TipcConnMsgSearch.
Caution	If TipcConnGetNumQueued returns FALSE, it does not store a value in <i>num_queued_return</i> .
See Also	TipcConnMsgNext, TipcConnMainLoop, TipcConnRead
Examples	This example prints the number of queued messages, then reads from a connection until at least three messages are available: <pre>T_INT4 num_queued; if (!TipcConnGetNumQueued(conn, &num_queued)) { return; /* error */ } TutOut("The connection has %d queued messages.\n", num_queued); while (num_queued < 3) { if (!TipcConnRead(conn, T_TIMEOUT_FOREVER)) { return; /* error */ } if (!TipcConnGetNumQueued(conn, &num_queued)) { return; /* error */ } }</pre>

TipcConnGetPid

Name	TipcConnGetPid — get the process ID of a connection's peer process
Synopsis	<pre>T_BOOL TipcConnGetPid(conn, pid_return) T_IPC_CONN conn; T_INT4 *pid_return;</pre>
Arguments	<i>conn</i> — connection from which to get process ID <i>pid_return</i> — storage for connection process ID
Return Values	TRUE if the process ID was successfully retrieved from the connection, FALSE otherwise.
Diagnostics	If TipcConnGetPid fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>pid_return</i> was null
Description	TipcConnGetPid gets the operating system process ID of a connection's peer process (the process with which this process is communicating through the connection). The peer process ID can be useful for diagnostic purposes.
Caution	If TipcConnGetPid returns FALSE, it does not store a value in <i>pid_return</i> .
See Also	TipcConnGetArch, TipcConnGetNode, TipcConnGetUniqueSubject, TipcConnGetUser
Examples	This example prints the process ID of a connection's peer process: <pre>T_INT4 pid; if (!TipcConnGetPid(conn, &pid)) { return; /* error */ } TutOut("Connection peer process ID is <%d>.\n", pid);</pre>

TipcConnGetSocket

Name TipcConnGetSocket — get the socket of a connection

Synopsis `T_BOOL TipcConnGetSocket(conn, socket_return)`
`T_IPC_CONN conn;`
`T_INT4 *socket_return;`

Arguments *conn* — connection from which to get socket
socket_return — storage for connection socket

Return Values TRUE if socket was successfully retrieved from connection, FALSE otherwise.

Diagnostics If TipcConnGetSocket fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *conn* was null or *socket_return* was null

Description TipcConnGetSocket gets the socket file descriptor of a connection. A socket is an operating system device which provides a communication link to another process. All data sent to and received from the connection is transmitted on this socket.

Caution If TipcConnGetSocket returns FALSE, it does not store a value in *socket_return*.

See Also TipcConnGetXtSource, TipcConnSetSocket

Examples This example prints the socket of a connection:

```
T_INT4 socket;

if (!TipcConnGetSocket(conn, &socket)) {
    return; /* error */
}

TutOut("Connection socket is <%d>.\n", socket);
```

TipcConnGetTimeout

Name	TipcConnGetTimeout — get a timeout property of a connection
Synopsis	<pre>T_BOOL TipcConnGetTimeout(<i>conn</i>, <i>timeout</i>, <i>value_return</i>) T_IPC_CONN <i>conn</i>; T_IPC_TIMEOUT <i>timeout</i>; T_REAL8 *<i>value_return</i>;</pre>
Arguments	<i>conn</i> — connection for which get timeout <i>timeout</i> — which timeout property to get <i>value_return</i> — storage for timeout value
Return Values	TRUE if the timeout property was successfully retrieved from connection, FALSE otherwise.
Diagnostics	If TipcConnGetTimeout fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>value_return</i> was null • T_ERR_VAL_INVALID — <i>timeout</i> was not a valid TIMEOUT value
Description	TipcConnGetTimeout gets a timeout property of a connection. The timeout properties control how often (in seconds) the connection is checked for possible network failures. The default for all timeout properties is 0.0, which means that timeouts are not checked. If <i>timeout</i> is T_IPC_TIMEOUT_READ, TipcConnGetTimeout gets the read timeout of a connection. TipcConnCheck uses the read timeout to control how often data is expected to be available for reading on the connection. A read timeout property of 0.0 disables checking for read timeouts. See TipcConnCheck for more information on how read timeouts are used. If <i>timeout</i> is T_IPC_TIMEOUT_WRITE, TipcConnGetTimeout gets the write timeout of a connection. TipcConnCheck uses the write timeout to control how often data is expected to be available to be written on the connection. A write timeout property of 0.0 disables checking for write timeouts. See TipcConnCheck for more information on how write timeouts are used. If <i>timeout</i> is T_IPC_TIMEOUT_KEEP_ALIVE, TipcConnGetTimeout gets the keep alive timeout of a connection. TipcConnKeepAlive uses the keep alive timeout to control how long to wait when checking if the other end of a connection is still alive. See TipcConnKeepAlive for more information on keep alives. A keep alive timeout property of 0.0 disables keep alives.

If *timeout* is `T_IPC_TIMEOUT_DELIVERY`, `TipcConnGetTimeout` gets the delivery timeout of a connection. `TipcConnCheck` and `TipcConnRead` use the delivery timeout to control how long to wait for guaranteed delivery of a message on the connection. A delivery timeout property of 0.0 disables checking for delivery timeouts. See `TipcConnCheck` and `TipcConnRead` for more information on how delivery timeouts are used.

Caution If `TipcConnGetTimeout` returns `FALSE`, it does not store a value in *value_return*.

See Also `TipcConnCheck`, `TipcConnFlush`, `TipcConnRead`, `TipcConnSetTimeout`

Examples This example gets the read timeout of a connection with `TipcConnGetTimeout` and prints it:

```
T_REAL8 timeout;

if (!TipcConnGetTimeout(conn, T_IPC_TIMEOUT_READ, &timeout)) {
    return; /* error */
}

TutOut("The connection read timeout is %s seconds.\n",
       TutRealToStr(timeout));
```


TipcConnGetUniqueSubject

Name	TipcConnGetUniqueSubject — get the unique subject of a connection's peer process
Synopsis	<pre>T_BOOL TipcConnGetUniqueSubject(conn, unique_subject_return) T_IPC_CONN conn; T_STR *unique_subject_return;</pre>
Arguments	<i>conn</i> — connection from which to get unique subject <i>unique_subject_return</i> — storage for connection unique subject
Return Values	TRUE if the unique subject was successfully retrieved from the connection, FALSE otherwise.
Diagnostics	If TipcConnGetUniqueSubject fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>conn</i> was null or <i>unique_subject_return</i> was null
Description	TipcConnGetUniqueSubject gets the unique subject of a connection's peer process (the process with which this process is communicating through the connection). The peer process unique subject can be useful for diagnostic purposes. The unique subject, which is used to uniquely identify a process, is stored in the option Unique_Subject.
Caution	If TipcConnGetUniqueSubject returns FALSE, it does not store a value in <i>unique_subject_return</i>. The value stored in <i>unique_subject_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcConnGetArch, TipcConnGetNode, TipcConnGetPid, TipcConnGetUser
Examples	This example prints the unique subject of a connection's peer process: <pre>T_STR unique_subject; if (!TipcConnGetUniqueSubject(conn, &unique_subject)) { return; /* error */ } TutOut("Connection peer process unique subject is <%s>.\n", unique_subject);</pre>

TipcConnGetUser

Name	TipcConnGetUser — get the user name of a connection's peer process
Synopsis	<pre>T_BOOL TipcConnGetUser(conn, user_return) T_IPC_CONN conn; T_STR *user_return;</pre>
Arguments	<i>conn</i> — connection to get user name from <i>user_return</i> — storage for connection user name
Return Values	TRUE if the user name was successfully retrieved from the connection, FALSE otherwise.
Diagnostics	If TipcConnGetUser fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>user_return</i> was null
Description	TipcConnGetUser gets the user name of a connection's peer process (the process that this process is communicating with over the connection). The peer process user name can be useful for diagnostic purposes.
Caution	If TipcConnGetUser returns FALSE, it does not store a value in <i>user_return</i>. The value stored in <i>user_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcConnGetArch, TipcConnGetNode, TipcConnGetPid, TipcConnGetUniqueSubject
Examples	This example prints the user name of a connection's peer process: <pre>T_STR user; if (!TipcConnGetUser(conn, &user)) { return; /* error */ } TutOut("Connection peer process user name is <%s>.\n", user);</pre>

TipcConnGetXtSource

Name	TipcConnGetXtSource — get source suitable for XtAppAddInput from a connection
Synopsis	<pre>T_BOOL TipcConnGetXtSource(conn, source_return) T_IPC_CONN conn; T_INT4 *source_return;</pre>
Arguments	<i>conn</i> — connection to get source from <i>source_return</i> — storage for source
Return Values	TRUE if source was successfully retrieved from connection, FALSE otherwise.
Diagnostics	If TipcConnGetXtSource fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>conn</i> was null or <i>source_return</i> was null
Description	TipcConnGetXtSource gets an XtAppAddInput-compatible source from a connection, which allows SmartSockets IPC to be mixed with Motif or any other Xt-based widget set. Both Xt and TipcConn have their own main loop functions: XtAppMainLoop for Xt and TipcConnMainLoop for TipcConn. Because only one main loop function can be used, it is easiest to use XtAppAddInput for incorporating other sources of input into Xt's event-handling mechanism. See the <i>TIBCO SmartSockets User's Guide</i> for more information on integrating SmartSockets with Motif/Xt. On UNIX and Windows, <i>source_return</i> is a socket file descriptor. On OpenVMS, <i>source_return</i> is an event flag in cluster zero, which is consistent with the OpenVMS implementation of XtAppAddInput.
Caution	If TipcConnGetXtSource returns FALSE, it does not store a value in <i>source_return</i> .
See Also	TipcConnGetSocket, TipcConnSetSocket

Examples This example shows how to call XtAppAddInput (on both UNIX and OpenVMS):

```
void my_xt_func(client_data, source, id)
XtPointer client_data; /* really (T_IPC_CONN) */
int *source;
XtInputId *id;
{
    T_IPC_CONN conn = (T_IPC_CONN)client_data;

    /* we know we have data to read from connection */
    if (!TipcConnMainLoop(conn, 0.0)) {
        return; /* error */
    }
} /* my_xt_func */

/* ===== */
/*...code from calling function is below */

XtAppContext app_context;
T_INT4 source;

if (!TipcConnGetXtSource(conn, &source) {
    return; /* error */
}

XtAppAddInput(app_context,
              (int)source,
#ifdef T_OS_VMS
              NULL,
#else
              (XtPointer)XtInputReadMask,
#endif
              my_xt_func,
              (XtPointer)conn);
```



Use TipcConnMainLoop, or TipcConnMsgNext in a loop, here instead of just calling TipcConnMsgNext once. Otherwise, messages might be read in but left in the connection message queue where XtAppMainLoop cannot see them. Timeout values other than 0.0 should also be used with caution here, as a non-zero timeout can cause the process to wait for data and thus decrease the responsiveness of the Motif/Xt user interface.

TipcConnGmdFileCreate

Name	TipcConnGmdFileCreate — create guaranteed message delivery area on a connection
Synopsis	<pre>T_BOOL TipcConnGmdFileCreate(conn) T_IPC_CONN conn;</pre>
Arguments	<i>conn</i> — connection to create GMD area for
Return Values	TRUE if the GMD area was successfully created for the connection, FALSE otherwise.
Diagnostics	If TipcConnGmdFileCreate fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null • T_ERR_ALREADY_EXISTS — the connection already has a GMD area open
Description	TipcConnGmdFileCreate creates a GMD area on a connection. A GMD area holds guaranteed message delivery information for both incoming and outgoing messages. There are two types of guaranteed message delivery: file-based GMD and memory-only GMD. File-based GMD stores GMD information in files to enable recovery if the program crashes and is restarted. File-based GMD is the default and also the recommended way to use GMD. Memory-only GMD stores GMD information in memory and thus is slightly faster than file-based GMD, but no recovery is possible if the program crashes and is restarted. You specify file-based or memory-based GMD by setting the <code>ipc_gmd_type</code> option to <code>default</code> or <code>memory</code>. If the option is set to <code>default</code>, file-based GMD is attempted. If it is unsuccessful, a warning is displayed and memory-based GMD is used. If the option is set to <code>memory</code>, only memory-based GMD is used, even if the <code>unique_subject</code> option is set. The default value of <code>ipc_gmd_type</code> is <code>default</code>. If TipcConnGmdFileCreate cannot create the files needed for file-based GMD, TipcConnGmdFileCreate prints a warning, switches to memory-only GMD, and creates a memory-only GMD area. To have full control over error recovery and GMD area creation, you may want to explicitly call TipcConnGmdFileCreate. You can use the function TipcGetGmdDir to verify that file-based GMD is configured properly.

For file-based GMD, `TipcConnGmdFileCreate` creates all files needed for guaranteed message delivery on a connection. For incoming messages a file with a `.hsn` file extension is used to save the highest sequence numbers received. For outgoing messages two files are used, one with a `.msg` file extension for copies of the messages and one with a `.map` file extension for locating the messages in the `.msg` file. These files are not designed for you to access them directly, but their names and purposes are documented here to illustrate how GMD works.

If the GMD files already exist, then `TipcConnGmdFileCreate` reads in the contents of the existing files. This allows a program to restart where it left off and resend any messages as well as detect redundant resent messages. To override this behavior, use `TipcConnGmdFileDelete` before `TipcConnGmdFileCreate` is called.

`TipcConnMsgSend` and `TipcConnRead` automatically call `TipcConnGmdFileCreate` if they handle a message for guaranteed delivery (that is, a message with a delivery mode of `T_IPC_DELIVERY_SOME` or `T_IPC_DELIVERY_ALL`) and the connection does not already have a GMD area open. The full pathnames of the GMD files are based on the values from `TipcGetGmdDir` and an internally generated base name. The value of the GMD base name is built from information similar to the logical connection names used by `RTclient` and `RTserver`. The GMD base name contains enough information to uniquely identify the connection persistently, so that if two processes have to recreate a connection because of a network failure, the GMD file names for the new connection match those of the old connection. The generated components of the pathnames are shrunk to 8 characters if they are too long for the native file system. (Windows Version 3.1 only allows 8-character file base names, and OpenVMS only allows 39-character file base names.)

For example, a server process with the unique subject `server_us` communicating with a client process with the unique subject `client_us` using the logical connection name `tcp:workstation1:2042` uses this full pathname:

UNIX:

```
/tmp/rtworks/gmd/server_us/tcp_workstation1_2042_client_us.hsn
```

OpenVMS:

```
dka300:[ssuser.gmd.server_us]tcp_workstation1_2042_client_us.hsn
```

Windows:

```
c:\temp\rtworks\gmd\server_us\tcp_workstation1_2042_client_us.hsn
```

These file names are long but necessary to uniquely identify the files in a persistent manner.

Each process that uses file-based GMD must properly configure the option `Unique_Subject` by setting it to a value other than the default `_Node_Pid`. The default `_Node_Pid` usually changes if the program is restarted (because of the new value for the process ID), which prevents the restarted program from finding the old GMD files.

Caution If `TipcConnGmdFileCreate` fails to create a file-based GMD area, a warning similar to this is output with `TutWarning`:

```
WARNING: Could not create GMD file(s) with base name
<local_node_test>
for connection <client:local:_node:test>.
The option Unique_Subject must be set to a value other
than the default (_workstation1_8668).
Switching to memory-only GMD.
```

The most common cause of `TipcConnGmdFileCreate` switching to memory-only GMD is not setting the option `Unique_Subject` to a value other than the default `_Node_Pid`. Each process that uses file-based GMD must properly configure `Unique_Subject`.

See Also `TipcConnGmdFileDelete`, `TipcConnMsgSend`, `TipcConnRead`

Examples This example creates a client connection to a server process, calls `TipcConnGmdFileCreate` to explicitly create a GMD area for the connection, and then sends a `NUMERIC_DATA` message with guaranteed delivery to the server process:

```
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_MSG msg;

/* have to set the Unique_Subject option to use file-based GMD */
TutCommandParseStr("setopt unique_subject example_client");

conn = TipcConnCreateClient("tcp:_node:4242");
if (conn == NULL) {
    return; /* error */
}
```

```

/* create GMD area before sending/receiving any messages */
if (!TipcConnGmdFileCreate(conn)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (!TipcConnMsgWrite(conn, mt,
                      T_IPC_PROP_DELIVERY_MODE, T_IPC_DELIVERY_ALL,
                      T_IPC_FT_STR, "test_value",
                      T_IPC_FT_REAL8, 10.0,
                      NULL)) {
    return; /* error */
}

```


TipcConnGmdFileDelete

Name	TipcConnGmdFileDelete — delete guaranteed message delivery files for a connection
Synopsis	<pre>T_BOOL TipcConnGmdFileDelete(conn) T_IPC_CONN conn;</pre>
Arguments	<i>conn</i> — connection for which to remove GMD files
Return Values	TRUE if the connection GMD files were successfully deleted from disk, FALSE otherwise.
Diagnostics	If TipcConnGmdFileDelete fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null • T_ERR_IN_USE — the connection GMD files are already open by this process and cannot be removed • T_ERR_C — a C error has occurred. Call the function TutErrNumGetC to obtain the related C-specific error value. • any error number from TipcGetGmdDir
Description	TipcConnGmdFileDelete deletes from disk guaranteed message delivery (GMD) files for a connection. TipcConnGmdFileDelete is useful for processes that wish to completely delete any obsolete GMD information. See TipcConnGmdFileCreate for more information on GMD areas. When a process using GMD creates a connection, one of two scenarios is possible: • the process does not care about GMD information from a previous connection with the same GMD configuration and should thus call TipcConnGmdFileDelete to delete the old files • the process is recreating the connection after a network failure and should thus call TipcConnGmdResend to resend all previously sent messages that may have been lost when the connection was destroyed TipcConnGmdFileDelete should be used carefully, as deleting GMD files removes all record of old messages that were supposed to be delivered.
Caution	Old GMD files must be removed before any messages with guaranteed delivery are sent or received on the connection. TipcConnGmdFileDelete does not delete the GMD files if they are already open by this process and in use for the connection. The GMD files are opened by TipcConnGmdFileCreate.

See Also TipcConnGmdFileCreate, TipcSrvGmdFileDelete

Examples This example creates a client connection to a server process, calls TipcConnGmdFileDelete to remove any existing GMD files for the connection, and then sends a NUMERIC_DATA message with guaranteed delivery to the server process:

```
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_MSG msg;

/* have to set the Unique_Subject option to use file-based GMD */
TutCommandParseStr("setopt unique_subject example_client");

conn = TipcConnCreateClient("tcp:_node:4242");
if (conn == NULL) {
    return; /* error */
}

/* Clear out old GMD files before sending/receiving any messages. Use TipcConnGmdResend
instead of TipcConnGmdFileDelete here if you wish to resend any previously sent messages with
GMD. */
if (!TipcConnGmdFileDelete(conn)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (!TipcConnMsgWrite(conn, mt,
                      T_IPC_PROP_DELIVERY_MODE, T_IPC_DELIVERY_ALL,
                      T_IPC_FT_STR, "test_value",
                      T_IPC_FT_REAL8, 10.0,
                      NULL)) {
    return; /* error */
}
```

TipcConnGmdMsgDelete

Name	TipcConnGmdMsgDelete — delete a message from GMD area after a GMD failure on a connection
Synopsis	<pre>T_BOOL TipcConnGmdMsgDelete(conn, msg) T_IPC_CONN conn; T_IPC_MSG msg;</pre>
Arguments	<i>conn</i> — connection from whose GMD area the message is to be deleted <i>msg</i> — message to delete
Return Values	TRUE if the guaranteed message was successfully deleted from the GMD area of the connection, FALSE otherwise.
Diagnostics	If TipcConnGmdMsgDelete fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — the connection is not set up for GMD (TipcConnGmdFileCreate must be called before TipcConnGmdMsgDelete) or the message was not previously sent with GMD • T_ERR_DOESNT_EXIST — <i>msg</i> could not be found in the GMD area for <i>conn</i> • T_ERR_C — a C error has occurred. Call the function TutErrNumGetC to obtain the related C-specific error value. This error number occurs if TipcConnGmdMsgDelete has a problem updating the GMD files.

Description TipcConnGmdMsgDelete deletes a message after a GMD failure on a connection. TipcConnGmdMsgDelete is intended to be used from a GMD_FAILURE connection process callback to terminate GMD for the message.

When a GMD_FAILURE message is processed by a sender, one of two scenarios is possible:

- the sender process does not want to resend the message, and thus uses TipcConnGmdMsgDelete to terminate GMD for the message
- the sender process does want to resend the message, and thus takes some kind of user-defined action to correct the problem and then uses TipcConnGmdMsgResend to resend the message

The standard callback TipcCbConnProcessGmdFailure uses TipcConnGmdMsgDelete to delete failed messages. See TipcCbConnProcessGmdFailure and TipcCbSrvProcessGmdFailure for more information on processing GMD_FAILURE messages.

Caution TipcConnGmdMsgDelete should only be used from a GMD_FAILURE process callback.

The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.

See Also TipcCbConnProcessGmdFailure, TipcCbSrvProcessGmdFailure, TipcConnGmdMsgResend, TipcSrvGmdMsgDelete

Examples This example creates a connection process callback for GMD_FAILURE messages which silently deletes the message that failed (this callback is essentially a silent version of TipcCbConnProcessGmdFailure). The existing callback TipcCbConnProcessGmdFailure is destroyed to avoid interfering with the new callback:

```
void T_ENTRY my_gmd_failure_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_IPC_MSG failed_msg;

    /* go to the first field of the GMD_FAILURE message */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }
}
```

```

    /* read the first field of the GMD_FAILURE message, which is */
    /* the message where GMD failed */
    if (!TipcMsgNextMsg(data->msg, &failed_msg))
        return; /* error */
    }

    /* terminate the sender's knowledge about the message for GMD */
    if (!TipcConnGmdMsgDelete(conn, failed_msg)) {
        return; /* error */
    }

} /* my_gmd_failure_cb */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;
T_CB cb;

mt = TipcMtLookupByNum(T_MT_GMD_FAILURE);
if (mt == NULL) {
    return; /* error */
}

/* create connection process callback for GMD_FAILURE messages */
if (TipcConnProcessCbCreate(conn, mt, my_gmd_failure_cb, NULL)
    == NULL) {
    return; /* error */
}

/* destroy standard callback for GMD_FAILURE messages */
cb = TipcConnProcessCbLookup(conn, mt,
                             TipcCbConnProcessGmdFailure, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}

```

TipcConnGmdMsgResend

Name	TipcConnGmdMsgResend — resend a message after a GMD failure on a connection
Synopsis	<pre>T_BOOL TipcConnGmdMsgResend(conn, msg) T_IPC_CONN conn; T_IPC_MSG msg;</pre>
Arguments	<i>conn</i> — connection on which to resend message <i>msg</i> — message to resend
Return Values	TRUE if the guaranteed message was successfully resent on the connection, FALSE otherwise.
Diagnostics	If TipcConnGmdMsgResend fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — the connection is not set up for GMD (TipcConnGmdFileCreate must be called before TipcConnGmdMsgResend) or the message was not previously sent with GMD
Description	TipcConnGmdMsgResend resends a messages after a GMD failure on a connection. TipcConnGmdMsgResend is intended to be used from a GMD_FAILURE connection process callback to resend a message with GMD. TipcConnGmdMsgResend preserves the sequence number property of the message and then calls TipcConnMsgSend to resend the message. The receiving processes automatically detect any redundant messages they have already processed. When a GMD_FAILURE message is processed by a sender, one of two scenarios is possible: • the sender process does not care to resend the message, and thus uses TipcConnGmdMsgDelete to terminate GMD for the message • the sender process does want to resend the message, and thus takes some kind of user-defined action to correct the problem and then uses TipcConnGmdMsgResend to resend the message See TipcCbConnProcessGmdFailure and TipcCbSrvProcessGmdFailure for more information on processing GMD_FAILURE messages.

Caution TipcConnGmdMsgResend should only be used from a GMD_FAILURE process callback.

If TipcConnMsgSend is used instead of TipcConnGmdMsgResend from a GMD_FAILURE process callback, then the message gets a new sequence number and the receiver may process the message twice.

TipcConnGmdMsgResend resends a message with GMD, but does not explicitly flush the message. See TipcConnGetAutoFlushSize for more information on message buffering.

The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.

See Also TipcCbConnProcessGmdFailure, TipcCbSrvProcessGmdFailure, TipcConnGmdMsgDelete

Examples This example creates a connection process callback for GMD_FAILURE messages which resends the message that failed. The existing callback TipcCbConnProcessGmdFailure is destroyed to avoid interfering with the new callback:

```
void T_ENTRY my_gmd_failure_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_IPC_MSG failed_msg;

    /* go to the first field of the GMD_FAILURE message */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* read the first field of the GMD_FAILURE message, which is */
    /* the message where GMD failed */
    if (!TipcMsgNextMsg(data->msg, &failed_msg))
        return; /* error */
    }

    /* at this point, a sender would typically take some */
    /* application-specific corrective action before resending */
    /* the message */
    take_corrective_actions();

    /* resend the message with GMD */
    if (!TipcConnGmdMsgResend(conn, failed_msg)) {
        return; /* error */
    }
} /* my_gmd_failure_cb */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;
T_CB cb;

mt = TipcMtLookupByNum(T_MT_GMD_FAILURE);
if (mt == NULL) {
    return; /* error */
}

/* create connection process callback for GMD_FAILURE messages */
if (TipcConnProcessCbCreate(conn, mt, my_gmd_failure_cb, NULL)
    == NULL) {
    return; /* error */
}

/* destroy standard callback for GMD_FAILURE messages */
cb = TipcConnProcessCbLookup(conn, mt,
                             TipcCbConnProcessGmdFailure, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}

```


TipcConnGmdResend

Name	TipcConnGmdResend — resend all guaranteed messages after a delivery failure on a connection
Synopsis	<pre>T_BOOL TipcConnGmdResend(conn) T_IPC_CONN conn;</pre>
Arguments	<i>conn</i> — connection to resend messages on
Return Values	TRUE if the guaranteed messages were successfully resent on the connection, FALSE otherwise.
Diagnostics	If TipcConnGmdResend fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null • any error number from TipcConnGmdFileCreate
Description	TipcConnGmdResend resends guaranteed messages after a delivery failure on a connection. TipcConnGmdResend reads all the messages from the GMD area and calls TipcConnMsgSend for each one to resend. The receiving processes automatically detect any redundant messages they have already processed. See TipcConnGmdFileCreate for more information on GMD areas. When a process using GMD creates a connection, one of two scenarios is possible: • the process does not care about GMD information from a previous connection with the same GMD configuration and should thus call TipcConnGmdFileDelete to delete the old files • the process is recreating the connection after a network failure and should thus call TipcConnGmdResend to resend all previously sent messages that may have been lost when the connection was destroyed If the connection does not have a GMD area open and there are existing GMD files on disk then TipcConnGmdResend calls TipcConnGmdFileCreate to open the existing GMD files.
Caution	TipcConnGmdResend should only be used when a connection is first created. TipcConnGmdResend resends all messages with GMD, but does not explicitly flush the messages. See TipcConnGetAutoFlushSize for more information on message buffering.
See Also	TipcConnGmdFileCreate

Examples This example creates a client connection to a server process, calls `TipcConnGmdResend` to resend any existing GMD files for the connection, then sends a `NUMERIC_DATA` message with guaranteed delivery to the server process:

```
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_MSG msg;

/* have to set the Unique_Subject option to use file-based GMD */
TutCommandParseStr("setopt unique_subject example_client");

conn = TipcConnCreateClient("tcp:_node:4242");
if (conn == NULL) {
    return; /* error */
}

/* Resend any previously sent messages with GMD. Use TipcConnGmdFileDelete instead of
TipcConnGmdResend here if you wish to delete the existing GMD files instead of resending any
previously sent GMD messages. */
if (!TipcConnGmdResend(conn)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (!TipcConnMsgWrite(conn, mt,
                      T_IPC_PROP_DELIVERY_MODE, T_IPC_DELIVERY_ALL,
                      T_IPC_FT_STR, "test_value",
                      T_IPC_FT_REAL8, 10.0,
                      NULL)) {
    return; /* error */
}
```

TipcConnKeepAlive

Name	TipcConnKeepAlive — check if the process at the other end of a connection is still alive
Synopsis	<pre>T_BOOL TipcConnKeepAlive(conn) T_IPC_CONN conn;</pre>
Arguments	<i>conn</i> — connection to initiate keep alive on
Return Values	TRUE if the other end of the connection responded in time, FALSE otherwise.
Diagnostics	If TipcConnKeepAlive fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null • T_ERR_IN_PROGRESS — TipcConnKeepAlive was called recursively for a connection (such as from a callback) • T_ERR_VAL_INVALID — the keep alive timeout of the connection is 0 (keep alives are disabled for the connection) • any error number from TipcConnMsgSendRpc
Description	TipcConnKeepAlive performs a remote procedure call to check if the process at the other end of a connection is still alive. This explicit attempt to check the health of a connection is called a keep alive. TipcConnKeepAlive is normally used only by TipcConnCheck, which automatically checks the connection based on the read timeout property of the connection. TipcConnKeepAlive can be called explicitly, though, to check the health of the connection. TipcConnKeepAlive performs these steps: 1. Temporarily sets the read timeout of the connection to 0. This is necessary to prevent recursive calls to TipcConnKeepAlive when the result message is being waited for. 2. Creates a KEEP_ALIVE_CALL message. 3. Sends the KEEP_ALIVE_CALL message with TipcConnMsgSend. 4. Flushes the message with TipcConnFlush. 5. Searches for KEEP_ALIVE_CALL and KEEP_ALIVE_RESULT messages with TipcConnMsgSearch. Any KEEP_ALIVE_CALL messages are processed with TipcConnMsgProcess immediately so that mutual keep alives from both ends of a connection are handled quickly. The search ends when the timeout period is reached or a KEEP_ALIVE_RESULT message is found.

6. Restores the read timeout of the connection and destroys the KEEP_ALIVE_CALL message.

If the other end of the connection is alive, it receives the KEEP_ALIVE_CALL message, processes it with TipcCbConnProcessKeepAliveCall, and sends a KEEP_ALIVE_RESULT message back to this end.

Caution If keep alives are used for non-RTserver connections, care must be taken to read and process messages at a regular interval. Otherwise the keep alives fail.

See Also TipcCbConnProcessKeepAliveCall, TipcConnGetTimeout, TipcConnSetTimeout, TipcConnMsgSendRpc

Examples This example shows a simple way to check the health of a connection every 10 seconds:

```
/* enable keep alives by setting the keep alive timeout to a non-zero value */
if (!TipcConnSetTimeout(conn, T_IPC_TIMEOUT_KEEP_ALIVE, 5.0)) {
    return; /* error */
}

for (;;) {
    if (!TipcConnKeepAlive(conn)) {
        return; /* error */
    }
    TutSleep(10.0);
}
```

Because TipcConnCheck automatically calls TipcConnKeepAlive when checking for reading, the above fragment could be rewritten as:

```
/* enable keep alives by setting the keep alive timeout to a non-zero value */
if (!TipcConnSetTimeout(conn, T_IPC_TIMEOUT_KEEP_ALIVE, 5.0)) {
    return; /* error */
}

/* enable read timeout checking by setting the read timeout to a non-zero value */
if (!TipcConnSetTimeout(conn, T_IPC_TIMEOUT_READ, 10.0)) {
    return; /* error */
}

/* based on the above timeout settings, TipcConnCheck will initiate a keep alive every 10 seconds
and wait for up to 5 seconds for the keep alive to complete */
if (!TipcConnCheck(conn, T_IO_CHECK_READ, T_TIMEOUT_FOREVER)) {
    return; /* error */
}
```

TipcConnLock

Name	TipcConnLock — acquire exclusive access to a connection
Synopsis	<pre>T_BOOL TipcConnLock(conn) T_IPC_CONN conn;</pre>
Arguments	<i>conn</i> — connection to lock
Return Values	TRUE if connection was successfully locked, FALSE if error occurred.
Diagnostics	If TipcConnLock fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR— <i>conn</i> was null • any error number from TutMutexLock
Description	TipcConnLock locks all of the internal synchronization objects within the connection. This ensures that the calling thread in a multi-threaded process has exclusive access to the connection. TipcConnLock may be called by a thread recursively.
Caution	Every call to TipcConnLock must be balanced by a call to TipcConnUnlock.
See Also	TipcConnUnlock
Examples	This example locks and unlocks a connection: <pre>if (!TipcConnLock(conn)) { return; /* error */ }</pre> <i>/* At this point, several TipcConn* functions could be called and there would be no risk of any other thread accessing the connection. The connection must be unlocked with TipcConnUnlock when exclusive access is no longer needed. */</i> <pre>if (!TipcConnUnlock(conn)) { return; /* error */ }</pre>

TipcConnMainLoop

Name	TipcConnMainLoop — read and process messages on a connection
Synopsis	<pre>T_BOOL TipcConnMainLoop(conn, timeout) T_IPC_CONN conn; T_REAL8 timeout;</pre>
Arguments	<i>conn</i> — connection to read and process messages on <i>timeout</i> — maximum number of seconds to wait for message to arrive
Return Values	TRUE if all pending messages were successfully processed on the connection, FALSE otherwise.
Diagnostics	If TipcConnMainLoop fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • any error number from TipcConnMsgNext except T_ERR_TIMEOUT_REACHED or T_ERR_WOULD_BLOCK. TipcConnMainLoop ignores these errors.
Description	TipcConnMainLoop reads and processes messages on a connection. TipcConnMainLoop is a convenience function that keeps calling TipcConnMsgNext with the time remaining from timeout until TipcConnMsgNext returns FALSE. For each message that TipcConnMainLoop gets, it processes the message with TipcConnMsgProcess and then destroys the message with TipcMsgDestroy. Use 0.0 for timeout to poll the connection and catch up on all pending messages that have accumulated. Use T_TIMEOUT_FOREVER for timeout to read and process messages indefinitely. As soon as TipcConnMsgNext returns FALSE, TipcConnMainLoop returns. If TipcConnMsgNext sets the global SmartSockets error number to T_ERR_TIMEOUT_REACHED (the timeout period was reached) or T_ERR_WOULD_BLOCK (the connection is in non-blocking mode with timeouts disabled), then TipcConnMainLoop returns TRUE; otherwise it returns FALSE. See TipcConnRead for more information on these error numbers.
Caution	None
See Also	TipcConnMsgNext, TipcConnMsgProcess

Examples This example does a time-consuming calculation, then catches up on its messages with TipcConnMainLoop:

```
perform_lengthy_calculation();

if (!TipcConnMainLoop(conn, 0.0)) {
    return; /* error */
}
```

TipcConnMsgInsert

Name	TipcConnMsgInsert — insert a message into queue of a connection
Synopsis	<pre>T_BOOL TipcConnMsgInsert(conn, msg, pos) T_IPC_CONN conn; T_IPC_MSG msg; T_INT4 pos;</pre>
Arguments	<i>conn</i> — connection to insert message into <i>msg</i> — message to insert <i>pos</i> — position to insert message
Return Values	TRUE if the message was successfully inserted into the queue of the connection, FALSE otherwise.
Diagnostics	If TipcConnMsgInsert fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — <i>pos</i> was negative or larger than the number of messages in the queue
Description	TipcConnMsgInsert inserts a message into the message queue of a connection. A connection has a priority queue of incoming messages. Use 0 for <i>pos</i> to insert the message as the first message in the queue, 1 for <i>pos</i> to insert the message as the second message, and so on. Use T_IPC_POS_PRIORITY for <i>pos</i> to insert the message in priority order, after all other message with equal or higher priority. TipcConnMsgInsert also calls the connection’s queue callbacks.
Caution	The message passed to TipcConnMsgInsert should not be destroyed by calling TipcMsgDestroy, as TipcConnMsgInsert takes responsibility for destroying the message. TipcConnMsgInsert does not make a copy of <i>msg</i>.
See Also	TipcConnMsgNext

Examples This example constructs a message and inserts it onto the front of a connection's message queue:

```
T_IPC_MT mt;
T_IPC_MSG msg;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "temperature", 98.6)) {
    return; /* error */
}

if (!TipcConnMsgInsert(conn, msg, 0)) {
    return; /* error */
}

/* At this point, msg "belongs" to the connection, and should */
/* not be destroyed by the caller */
```

TipcConnMsgNext

Name	TipcConnMsgNext — get the next message from a connection
Synopsis	<pre>T_IPC_MSG TipcConnMsgNext(<i>conn</i>, <i>timeout</i>) T_IPC_CONN <i>conn</i>; T_REAL8 <i>timeout</i>;</pre>
Arguments	<i>conn</i> — connection to get next message from <i>timeout</i> — maximum number of seconds to wait for message to arrive
Return Values	A message if successful, NULL otherwise.
Diagnostics	If TipcConnMsgNext fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • any error number from TipcConnRead
Description	TipcConnMsgNext gets the next message from a connection. A connection has a priority queue of incoming messages. TipcConnMsgNext removes the message from the front of the queue (which is usually the message with the highest priority) and calls the connection’s queue callbacks. As a convenience, TipcConnMsgNext sets the current field of the returned message to field number 0 (the first field) with TipcMsgSetCurrent. To get a message other than the first message, use TipcConnMsgSearch. Once a message is acquired by calling TipcConnMsgNext, it is normally processed with TipcConnMsgProcess and then destroyed by calling TipcMsgDestroy. If the connection’s message queue is not empty, TipcConnMsgNext can return a message immediately. If the message queue is empty, TipcConnMsgNext keeps calling TipcConnRead with timeout until at least one full message is read or TipcConnRead returns FALSE. Use 0.0 for timeout to poll for a message without blocking. Use T_TIMEOUT_FOREVER for timeout to wait until a message arrives.

Caution The message returned by `TipcConnMsgNext` should be destroyed when it is no longer needed by calling `TipcMsgDestroy`, unless the message is passed to another function which takes responsibility for destroying the message (such as `TipcConnMsgInsert`).

For performance reasons, if `TipcConnMsgNext` has to call `TipcConnRead` several times, `TipcConnMsgNext` does not calculate how much of timeout has elapsed. Instead it calls `TipcConnRead` with the full timeout each time. If a large message is arriving very slowly in small increments, then the elapsed time may be much longer than the desired timeout.

See Also `TipcConnMsgProcess`, `TipcConnMainLoop`, `TipcConnRead`

Examples This example waits for up to 10 seconds to get the next message, processes the message, and finally destroys the message:

```
T_IPC_MSG msg;

msg = TipcConnMsgNext(conn, 10.0);
if (msg == NULL) {
    return; /* error */
}

if (!TipcConnMsgProcess(conn, msg)) {
    return; /* error */
}

if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcConnMsgProcess

Name	TipcConnMsgProcess — process a message in a connection
Synopsis	<pre>T_BOOL TipcConnMsgProcess(conn, msg) T_IPC_CONN conn; T_IPC_MSG msg;</pre>
Arguments	<i>conn</i> — connection to process message in <i>msg</i> — message to be processed
Return Values	TRUE if message was successfully processed, FALSE otherwise.
Diagnostics	If TipcConnMsgProcess fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcConnMsgProcess processes a message in a connection by calling the process callbacks or default callbacks. The default callbacks are only called if there are no non-global process callbacks for the type of message being processed. Processing a message with two levels of callbacks allows for more flexibility. For example, RTserver creates a few process callbacks for the message types it needs to process locally (such as SUBJECT_SET_SUBSCRIBE), but the majority of the message types (such as NUMERIC_DATA) are simply routed to their destination by a default callback. Any message can be processed by TipcConnMsgProcess, not just messages read from a connection. For example, a generic message file data source could be implemented which read messages from a file with TipcMsgFileRead and then processed those same messages with TipcConnMsgProcess.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnDefaultCbCreate, TipcConnMsgNext, TipcConnMainLoop, TipcConnProcessCbCreate

Examples This example gets the next message from a connection, processes it with `TipcConnMsgProcess`, and finally destroys the message:

```
msg = TipcConnMsgNext(conn, T_TIMEOUT_FOREVER);
if (msg == NULL) {
    return; /* error */
}

if (!TipcConnMsgProcess(conn, msg)) {
    return; /* error */
}

if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcConnMsgSearch

Name	TipcConnMsgSearch — search the message queue of a connection for a specific message
Synopsis	<pre>T_IPC_MSG TipcConnMsgSearch(<i>conn</i>, <i>timeout</i>, <i>func</i>, <i>arg</i>) T_IPC_CONN <i>conn</i>; T_REAL8 <i>timeout</i>; T_IPC_CONN_MSG_SEARCH_FUNC <i>func</i>; T_PTR <i>arg</i>;</pre>
Arguments	<i>conn</i> — connection to search <i>timeout</i> — maximum number of seconds to wait for desired message to arrive <i>func</i> — function to call once for each message <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	A message if successful, NULL otherwise.
Diagnostics	If TipcConnMsgSearch fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_TIMEOUT_REACHED — the <i>timeout</i> period was reached • any error number from TipcConnRead
Description	TipcConnMsgSearch searches the message queue of a connection for a specific message. A connection has a priority queue of incoming messages. TipcConnMsgSearch traverses the messages in the queue and calls <i>func</i> once for each message. When <i>func</i> returns TRUE, TipcConnMsgSearch removes the message from the queue, calls the connection’s queue callbacks, and returns the message. If TipcConnMsgSearch reaches the end of the queue, it calls TipcConnRead with the time remaining from timeout, and searches the queue again. TipcConnMsgSearch continues reading and searching until <i>func</i> returns TRUE, TipcConnRead returns FALSE, or the timeout is reached. As a convenience, TipcConnMsgSearch sets the current field of the returned message to field number 0 (the first field) with TipcMsgSetCurrent. TipcConnMsgSearch is useful for processing messages out of priority order, such as when waiting for the result from a remote procedure call or when checking for a message of a specific type. To get the first message from the message queue, use TipcConnMsgNext instead. Once a message is acquired by calling TipcConnMsgSearch, it is normally processed (either with TipcConnMsgProcess

or by accessing it directly without any callbacks) and then destroyed by calling `TipcMsgDestroy`. Use 0.0 for timeout to search for a message without blocking. Use `T_TIMEOUT_FOREVER` for timeout to wait until a message arrives. To search for a message with a specific type, use the convenience function `TipcConnMsgSearchType`.

Caution The message returned by `TipcConnMsgSearch` should be destroyed when it is no longer needed by calling `TipcMsgDestroy`, unless the message is passed to another function which takes responsibility for destroying the message (such as `TipcConnMsgInsert`).

The `T_ENTRY` declaration specifier is required in the definition of all callback functions as well as their prototypes.

See Also `TipcConnMsgProcess`, `TipcConnMsgSearchType`, `TipcConnRead`

Examples Remote procedure calls (RPCs) can be easily implemented between two ends of a connection by the caller end sending one message as the call and other end sending one message back as the result. This example calculates the tangent of a number with such an RPC. It sends a call message and waits for up to 10 seconds for a result message to be sent back from the other end of the connection. User code would normally use `TipcConnMsgSendRpc` (or even just `TipcConnMsgSearchType`) to implement this, but this example uses `TipcConnMsgSearch` to show the functionality in more detail. Shown first is the caller fragment:

```
#define USER_MT_TANGENT_CALL 100
#define USER_MT_TANGENT_RESULT 101

/* search function */
static T_BOOL T_ENTRY search_for_result(conn, msg, arg)
T_IPC_CONN conn;
T_IPC_MSG msg;
T_PTR arg; /* not used */
{
    T_BOOL status;
    T_IPC_MT mt;
    T_INT4 num;

    if (!TipcMsgGetType(msg, &mt)) {
        return; /* error */
    }
    if (!TipcMtGetNum(mt, &num)) {
        return; /* error */
    }
}
```

```

    /* return TRUE if message type matches what we want */
    return num == USER_MT_TANGENT_RESULT;
} /* search_for_result */
/* calculate tangent via RPC */
T_REAL8 tangent_rpc(num)
T_REAL8 num;
{
    T_IPC_MT mt;
    T_IPC_MSG msg;
    T_REAL8 result;

    mt = TipcMtLookupByNum(USER_MT_TANGENT_CALL);
    if (mt == NULL) {
        return; /* error */
    }

    if (!TipcConnMsgWrite(conn, mt,
                          T_IPC_FT_REAL8, num,
                          NULL)) {
        return; /* error */
    }

    /* assume auto flush size is not T_IPC_NO_AUTO_FLUSH, thus */
    /* no flush needed */

    /* search for result message */
    msg = TipcConnMsgSearch(conn, 10.0, search_for_result, NULL);
    if (msg == NULL) {
        return; /* error */
    }

    /* process result without using TipcConnMsgProcess */

    if (!TipcMsgNextReal8(msg, &result)) {
        return; /* error */
    }
    if (!TipcMsgDestroy(msg)) {
        return; /* error */
    }
    return result;
} /* tangent_rpc */

```

This is the called fragment:

```

/* callback to process RPC "call" message */
static T_ENTRY void process_call(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_REAL8 num;
    T_IPC_MT mt;

```



```

/* have to start at first field */
if (!TipcMsgSetCurrent(data->msg, 0)) {
    return; /* error */
}

/* access the RPC call information */
if (!TipcMsgNextReal8(data->msg, &num)) {
    return; /* error */
}

/* send a result message back to the caller */
mt = TipcMtLookupByNum(USER_MT_TANGENT_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (!TipcConnMsgWrite(conn, mt,
                      T_IPC_FT_REAL8, tan(num),
                      NULL)) {
    return; /* error */
}

if (!TipcConnFlush(conn)) {
    return; /* error */
}
} /* process_call */

/* ===== */
/*...code from calling function is below */

/* fragment that uses process_call */
T_IPC_MT mt;

mt = TipcMtLookupByNum(USER_MT_TANGENT_CALL);
if (mt == NULL) {
    return; /* error */
}

if (TipcConnProcessCbCreate(conn, mt, process_call, NULL) == NULL)
{
    return; /* error */
}
TipcConnMainLoop(conn, T_TIMEOUT_FOREVER); /* process call messages */

```

TipcConnMsgSearchType

Name	TipcConnMsgSearchType — search the message queue of a connection for a message with a specific message type
Synopsis	<pre>T_IPC_MSG TipcConnMsgSearchType(conn, timeout, mt) T_IPC_CONN conn; T_REAL8 timeout; T_IPC_MT mt;</pre>
Arguments	<i>conn</i> — connection to search <i>timeout</i> — maximum number of seconds to wait for desired message to arrive <i>mt</i> — desired message type
Return Values	A message if successful, NULL otherwise.
Diagnostics	If TipcConnMsgSearchType fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>mt</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • any error number from TipcConnMsgSearch
Description	TipcConnMsgSearchType is a convenience function that searches for a message with a specific type by calling TipcConnMsgSearch. The first message found in the connection's message queue with that type is removed from the queue and returned.
Caution	Unless you pass the message returned by TipcConnMsgSearchType to another function that takes responsibility for destroying it (such as TipcConnMsgInsert), when the message is no longer needed destroy it by calling TipcMsgDestroy.
See Also	TipcConnMsgSearch
Examples	This code fragment waits up to 10 seconds for a CONTROL message to arrive from the other end of a connection:

```
mt = TipcMtLookupByNum(T_MT_CONTROL);
if (mt == NULL) {
    return; /* error */
}
msg = TipcConnMsgSearchType(conn, 10.0, mt);
if (msg == NULL) {
    return; /* error */
}
```

TipcConnMsgSend

Name	TipcConnMsgSend — send a message through a connection
Synopsis	<pre>T_BOOL TipcConnMsgSend(conn, msg) T_IPC_CONN conn; T_IPC_MSG msg;</pre>
Arguments	<i>conn</i> — connection to send message through <i>msg</i> — message to be sent
Return Values	TRUE if message was successfully sent, FALSE otherwise.
Diagnostics	If TipcConnMsgSend fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_LARGE — the connection GMD area maximum size was exceeded • T_ERR_C — a C error has occurred. Call the function TutErrNumGetC to obtain the related C-specific error value. This error indicates that a copy of the message could not be saved to the connection GMD area. • any error number from TipcConnFlush
Description	TipcConnMsgSend sends a message through a connection. The connection's write callbacks are first called for the message. If the message has a delivery mode of either T_IPC_DELIVERY_SOME or T_IPC_DELIVERY_ALL, then TipcConnMsgSend performs some extra steps for guaranteed message delivery. If the connection does not have a GMD area open, then TipcConnMsgSend calls TipcConnGmdFileCreate to create the necessary GMD area. TipcConnMsgSend increments an internal per-connection sequence number, sets the sequence number of the message to the incremented value, and then saves a copy of the message to the GMD area so that the message can be easily resent. With file-based GMD, recovery can take place even if this process crashes and has to be restarted. See TipcConnGmdFileCreate for more information on creation of GMD areas. If the message delivery timeout property is not set, then the connection delivery timeout value is used for GMD.

TipcConnMsgSend then converts the message into a message packet (checking the message header string encode property to see if header strings should be encoded as integers to save space), and appends the message packet to the connection's write buffer. TipcConnMsgSend then calls TipcConnFlush if the number of bytes in the connection's write buffer is greater than the connection's auto flush size.

Caution TipcConnMsgSend does not set the sender property or destination property of the message.

Use TipcSrvMsgSend or TipcSrvConnMsgSend, not TipcConnMsgSend, to send messages to RTserver.

If TipcConnGmdFileCreate fails to create a file-based GMD area, a warning similar to this is output with TutWarning:

```
WARNING: Could not create GMD file(s) with base name
<local_node_test>
for connection <client:local:_node:test>.
The option Unique_Subject must be set to a value other
than the default (_workstation1_8668).
Switching to memory-only GMD.
```

If the above warning appears, the process is not configured properly to use file-based GMD. See TipcConnGmdFileCreate for more information on creating GMD areas.

See Also TipcConnFlush, TipcConnMsgWrite, TipcConnRead, TipcSrvMsgSend, TipcSrvConnMsgSend

Examples This example creates a message, appends fields, and sends the message with TipcConnMsgSend:

```
mt = TipcMtLookupByNum(T_MT_INFO);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStr(msg, "Open the pod bay doors, Hal")) {
    return; /* error */
}

/*
Normally the message sender and destination would be set here.
The sender/destination are very connection-specific.
*/
```

```
/* send the message */
if (!TipcConnMsgSend(conn, msg)) {
    return; /* error */
}
if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcConnMsgSendRpc

Name	TipcConnMsgSendRpc — make a remote procedure call (RPC) with messages on a connection
Synopsis	<pre>T_IPC_MSG TipcConnMsgSendRpc(conn, call_msg, timeout) T_IPC_CONN conn; T_IPC_MSG call_msg; T_REAL8 timeout;</pre>
Arguments	<i>conn</i> — connection to make RPC on <i>call_msg</i> — call message for RPC <i>timeout</i> — maximum number of seconds to wait for result message to arrive
Return Values	A result message if successful, NULL otherwise.
Diagnostics	If TipcConnMsgSendRpc fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>call_msg</i> was null • T_ERR_MSG_INVALID — <i>call_msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_TIMEOUT_REACHED — the <i>timeout</i> period was reached • any error number from TipcMtLookupByNum • any error number from TipcConnMsgSend • any error number from TipcConnFlush • any error number from TipcConnMsgSearchType
Description	TipcConnMsgSendRpc makes a remote procedure call (RPC) with messages on a connection. One message is sent as the RPC call from the caller end of the connection, and another message is sent back as the RPC result to the caller. The callee end of the connection must be prepared to receive the call message and send back the result message. The relationship between the call and result message is that the message type number of the result message is always one greater than the message type number of the call message. TipcConnMsgSendRpc sends the message with TipcConnMsgSend, flushes all pending outgoing data with TipcConnFlush, and then waits for the result with TipcConnMsgSearchType.

A message returned by `TipcConnMsgSendRpc` is normally processed (either with `TipcConnMsgProcess`, or by accessing it directly without any callbacks) and then destroyed by calling `TipcMsgDestroy`. Use `T_TIMEOUT_FOREVER` for timeout to wait until a message arrives.

Caution The message returned by `TipcConnMsgSendRpc` should be destroyed when it is no longer needed by calling `TipcMsgDestroy`, unless the message is passed to another function which takes responsibility for destroying the message, such as `TipcConnMsgInsert`.

The message type number of the result message must be one greater than the message type number of the call message.

If both ends of the connection attempt to make an RPC at the same time, a deadlock condition can occur.

See Also `TipcConnMsgProcess`, `TipcConnMsgSearchType`

Examples This example calculates the tangent of a number with a message-based connection RPC. It sends a call message and waits for up to 10 seconds for a result message to be sent back from the other end of the connection. Shown first is a utility fragment that must be used by both the caller and the called program:

```
#define USER_MT_TANGENT_CALL 100
#define USER_MT_TANGENT_RESULT 101 /* call num plus one */

/* utility function to create message types */
static void create_tangent_mt()
{
    T_IPC_MT mt;

    mt = TipcMtCreate("tangent_call", USER_MT_TANGENT_CALL,
                     "real8");
    if (mt == NULL) {
        return; /* error */
    }

    mt = TipcMtCreate("tangent_result", USER_MT_TANGENT_RESULT,
                     "real8");
    if (mt == NULL) {
        return; /* error */
    }
} /* create_tangent_mt */
```

Shown next is the caller fragment:

```

/* calculate tangent via RPC */
T_REAL8 tangent_rpc(num)
T_REAL8 num;
{
    T_IPC_MT mt;
    T_IPC_MSG call_msg;
    T_IPC_MSG result_msg;
    T_REAL8 result;

    mt = TipcMtLookupByNum(USER_MT_TANGENT_CALL);
    if (mt == NULL) {
        return; /* error */
    }

    call_msg = TipcMsgCreate(mt);
    if (call_msg == NULL) {
        return; /* error */
    }

    if (!TipcMsgAppendReal8(call_msg, num)) {
        return; /* error */
    }

    /* send/recv RPC messages */
    result_msg = TipcConnMsgSendRpc(conn, call_msg, 10.0);
    if (result_msg == NULL) {
        return; /* error */
    }

    /* process result without using TipcConnMsgProcess */

    if (!TipcMsgNextReal8(result_msg, &result)) {
        return; /* error */
    }
    if (!TipcMsgDestroy(result_msg)) {
        return; /* error */
    }
    if (!TipcMsgDestroy(call_msg)) {
        return; /* error */
    }
    return result;
} /* tangent_rpc */

```

This is the called program's fragment:

```

/* callback to process RPC "call" message */
static void T_ENTRY process_call(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_REAL8 num;
    T_IPC_MT mt;

```



```

/* have to start at first field */
if (!TipcMsgSetCurrent(data->msg, 0)) {
    return; /* error */
}

/* access the RPC call information */
if (!TipcMsgNextReal8(data->msg, &num)) {
    return; /* error */
}

mt = TipcMtLookupByNum(USER_MT_TANGENT_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* send a result message back to the caller */
if (!TipcConnMsgWrite(conn, mt,
                      T_IPC_FT_REAL8, tan(num),
                      NULL)) {
    return; /* error */
}
if (!TipcConnFlush(conn)) {
    return; /* error */
}
} /* process_call */

/* ===== */
/*...code from calling function is below */

/* fragment that uses process_call */
T_IPC_MT mt;

mt = TipcMtLookupByNum(USER_MT_TANGENT_CALL);
if (mt == NULL) {
    return; /* error */
}

if (TipcConnProcessCbCreate(conn, mt, process_call, NULL) == NULL)
{
    return; /* error */
}

TipcConnMainLoop(conn, T_TIMEOUT_FOREVER); /* process call messages */

```

TipcConnMsgWrite

Name	TipcConnMsgWrite — construct a message and send it through a connection
Synopsis	<pre>T_BOOL TipcConnMsgWrite(conn, mt, ...) T_IPC_CONN conn; T_IPC_MT mt;</pre>
Arguments	<i>conn</i> — connection to send message through <i>mt</i> — message type additional arguments — null-terminated field type-value pairs
Return Values	TRUE if the message was successfully constructed and sent, FALSE otherwise.
Diagnostics	If TipcConnMsgWrite fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>mt</i> was null • any error number from TipcConnMsgWriteVa
Description	TipcConnMsgWrite is a convenience function that takes a null-terminated variable number of field type-value pairs, creates a message, appends these fields to the message, sends the message, and destroys the message. The field type part of the pair is a T_IPC_FT and the value part of the pair corresponds to one of these arguments: • The argument for the related TipcMsgAppend* function. For example, if the type part is T_IPC_FT_REAL8, then the value part is a REAL8 number. • The argument for the related TipcMsgSet* function. For example, if the type part is T_IPC_PROP_DELIVERY_MODE, then the value part is a T_IPC_DELIVERY_MODE setting. TipcConnMsgWrite is a short wrapper function that calls TipcConnMsgWriteVa.
Caution	The last argument passed to TipcConnMsgWrite must be NULL; otherwise TipcConnMsgWrite may append garbage fields to the message. Using TipcConnMsgWrite is slightly slower than using the TipcMsgAppend functions directly.
See Also	TipcConnMsgWriteVa

Examples This example sends a message with TipcConnMsgWrite:

```
mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (!TipcConnMsgWrite(conn, mt,
                      T_IPC_FT_STR, "speed_limit",
                      T_IPC_FT_REAL8, 55.0,
                      NULL)) {
    return; /* error */
}
```

TipcConnMsgWriteVa

Name	TipcConnMsgWriteVa — construct a message and send it through a connection (va_list version)
Synopsis	<pre>T_BOOL TipcConnMsgWriteVa(conn, mt, var_arg_list) T_IPC_CONN conn; T_IPC_MT mt; va_list var_arg_list;</pre>
Arguments	<i>conn</i> — connection to send message through <i>mt</i> — message type <i>var_arg_list</i> — argument list
Return Values	TRUE if the message was successfully constructed and sent, FALSE otherwise.
Diagnostics	If TipcConnMsgWriteVa fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>mt</i> was null • any error number from TipcMsgWriteVa • any error number from TipcConnMsgSend
Description	TipcConnMsgWriteVa is the non-varargs version of TipcConnMsgWrite (TipcConnMsgWrite actually calls TipcConnMsgWriteVa). TipcConnMsgWriteVa can be used to implement a varargs function. In the C language, a varargs function (one that takes a variable number of arguments) cannot call another varargs function with the original arguments. A varargs function usually calls a helper function that takes a fixed number of arguments, with the last argument usually of type va_list. TipcConnMsgWriteVa is TipcConnMsgWrite’s helper function.
Caution	None
See Also	TipcConnMsgWrite, TipcConnMsgWriteVa

Examples This example shows how to write a varargs function in ANSI C that constructs a message and sends it on a predefined connection:

```
extern T_IPC_CONN my_conn;

T_BOOL my_conn_msg_write(T_IPC_MT mt,
                        ...)
{
    va_list var_arg_list;
    T_BOOL status;

    va_start(var_arg_list, mt);
    status = TipcConnMsgWriteVa(my_conn, mt, var_arg_list);
    va_end(var_arg_list);
    return status;
} /* my_conn_msg_write */
```

TipcConnProcessCbCreate

Name	TipcConnProcessCbCreate — create a process callback in a connection
Synopsis	<pre>T_CB TipcConnProcessCbCreate(conn, mt, func, arg) T_IPC_CONN conn; T_IPC_MT mt; T_IPC_CONN_PROCESS_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection to create callback for <i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcConnProcessCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcConnProcessCbCreate creates a process callback in a connection. These callbacks are called by TipcConnMsgProcess to perform the normal processing of a message, such as storing numeric data values and then acting on the variables. See the <i>TIBCO SmartSockets Utilities</i> for more information on callbacks. A process callback is usually created for a specific message type and connection. If the message type is null, then a global process callback is created for all message types on that connection.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnDefaultCbCreate, TipcConnMsgProcess, TipcConnProcessCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example creates a global connection process callback that writes all processed messages to a message file:

```
void T_ENTRY my_conn_process_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg; /* really (T_IPC_MSG_FILE) */
{
    T_IPC_MSG msg = data->msg;
    T_IPC_MSG_FILE msg_file = arg;

    if (!TipcMsgFileWrite(msg_file, msg)) {
        return; /* error */
    }
} /* my_conn_process_cb */

/* ===== */
/*...code from calling function is below */

msg_file = TipcMsgFileCreate("my_log.msg",
                             T_IPC_MSG_FILE_CREATE_WRITE);
if (msg_file == NULL) {
    return; /* error */
}

/* global connection process callback */
if (TipcConnProcessCbCreate(conn, NULL, my_conn_process_cb,
                           msg_file) == NULL) {
    return; /* error */
}
```

TipcConnProcessCbLookup

Name	TipcConnProcessCbLookup — look up a process callback in a connection
Synopsis	<pre>T_CB TipcConnProcessCbLookup(conn, mt, func, arg) T_IPC_CONN conn; T_IPC_MT mt; T_IPC_CONN_PROCESS_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection to look up callback for <i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcConnProcessCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcConnProcessCbLookup looks up a process callback with the desired message type, function, and argument in a connection. These callbacks are called by TipcConnMsgProcess to perform the normal processing of a message, such as storing numeric data values and then acting on the slots. See TipcConnProcessCbCreate for more information on connection process callbacks.
Caution	None
See Also	TipcConnProcessCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example looks up a NUMERIC_DATA connection process callback and a global connection process callback:

```
mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

cb = TipcConnProcessCbLookup(conn, mt, my_conn_process_cb, NULL);
if (cb == NULL) {
    return; /* error */
}

/* global connection process callback */
cb = TipcConnProcessCbLookup(conn, NULL, my_global_process_cb,
NULL);
if (cb == NULL) {
    return; /* error */
}
```

TipcConnQueueCbCreate

Name	TipcConnQueueCbCreate — create a queue callback in a connection
Synopsis	<pre>T_CB TipcConnQueueCbCreate(conn, mt, func, arg) T_IPC_CONN conn; T_IPC_MT mt; T_IPC_CONN_QUEUE_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection to create callback for <i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcConnQueueCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcConnQueueCbCreate creates a queue callback in a connection. These callbacks are called when a message is inserted into or deleted from a connection’s queue of incoming messages. Connection queue callbacks are useful for watching the messages that have been read in by a connection, but not yet processed. There is some overlap between read callbacks and queue callbacks. See <i>TIBCO SmartSockets Utilities</i> for more information on callbacks. A queue callback is usually created for a specific message type and connection. If the message type is null, then a global queue callback is created for all message types on that connection.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnQueueCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example creates a global connection queue callback that prints out how the queue is changing:

```
void T_ENTRY my_conn_queue_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_QUEUE_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    if (data->insert_flag)
        TutOut("A message has been inserted at position %d\n",
            data->position);
    else
        TutOut("A message has been deleted at position %d\n",
            data->position);
} /* my_conn_queue_cb */

/* ===== */
/*...code from calling function is below */

/* global connection queue callback */
if (TipcConnQueueCbCreate(conn, NULL, my_conn_queue_cb, NULL)
    == NULL) {
    return; /* error */
}
```

TipcConnQueueCbLookup

Name	TipcConnQueueCbLookup — look up a queue callback in a connection
Synopsis	<pre>T_CB TipcConnQueueCbLookup(conn, mt, func, arg) T_IPC_CONN conn; T_IPC_MT mt; T_IPC_CONN_QUEUE_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection to look up callback for <i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcConnQueueCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcConnQueueCbLookup looks up a connection queue callback with the desired message type, function, and argument in a connection. These callbacks are called when a message is inserted into or deleted from a connection’s queue of incoming messages. See TipcConnQueueCbCreate for more information on connection queue callbacks.
Caution	None
See Also	TipcConnQueueCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example looks up a NUMERIC_DATA connection queue callback and a global connection queue callback:

```
mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

cb = TipcConnQueueCbLookup(conn, mt, my_conn_queue_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
/* global connection queue callback */
cb = TipcConnQueueCbLookup(conn, NULL, my_global_queue_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
```

TipcConnRead

Name TipcConnRead — read data from a connection and queue messages in priority order

Synopsis `T_BOOL TipcConnRead(conn, timeout)`
`T_IPC_CONN conn;`
`T_REAL8 timeout;`

Arguments *conn* — connection to read from
timeout — maximum number of seconds to wait for data to arrive

Return Values TRUE if data was successfully read from connection, FALSE otherwise.

Diagnostics If TipcConnRead fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *conn* was null
- T_ERR_VAL_TOO_SMALL — *timeout* was negative
- T_ERR_IN_PROGRESS — TipcConnRead was called recursively for a connection (from a read callback or queue callback)
- T_ERR_C — a C error has occurred. Call the function TutErrNumGetC to obtain the related C-specific error value.
- T_ERR_OS — an operating system error has occurred. Call the function TutErrNumGetOs to obtain the related OS-specific error value.
- T_ERR_SOCKET — a socket error has occurred. Call the function TutErrNumGetSocket to obtain the related socket-specific error value.
- T_ERR_EOF — the other end of the connection’s socket has been closed (EOF has been reached). The connection should be destroyed and then a new connection created if necessary.
- T_ERR_WOULD_BLOCK — *conn* is in non-blocking mode, and no data could be received without blocking
- any error number from TipcConnFlush
- any error number from TipcConnCheck

Description TipcConnRead reads and queues messages that arrive (within a certain period of time) from a connection’s socket. If the connection’s auto flush size is not T_IPC_NO_AUTO_FLUSH, then TipcConnRead first calls TipcConnFlush to flush the connection’s write buffer (so that any responses to the outgoing data are available sooner). TipcConnRead then reads data (in the form of message packets)

from a connection's socket and calls the connection decode callbacks. If necessary, the integers and strings in the message header are converted to the formats used by the receiving process. TipcConnRead then converts the message packets into messages, and calls the connection's read callbacks. TipcConnRead returns as soon as at least one message is read, so that high-data-rate applications do not sit forever reading data.

For each message read in, if the message has a delivery mode of either `T_IPC_DELIVERY_SOME` or `T_IPC_DELIVERY_ALL` then TipcConnRead performs some extra steps for guaranteed message delivery. If the connection does not have a GMD area open, then TipcConnRead calls TipcConnGmdFileCreate to create the necessary GMD area. TipcConnRead checks if the message has already been received and acknowledged, and, if so, immediately reacknowledges the message and destroys the message. TipcConnRead also directly acts on incoming acknowledgment messages that are received for messages that have been sent on this connection.

If any messages are read in, TipcConnRead checks for guaranteed message delivery timeouts. When messages with a delivery mode of `T_IPC_DELIVERY_SOME` or `T_IPC_DELIVERY_ALL` are sent through a connection with TipcConnMsgSend, copies of the messages are saved in the connection GMD area. The copies are removed when acknowledgment of delivery is received by the sender from the receiving process(es). TipcConnRead checks how many seconds have elapsed since each previously sent guaranteed message was sent. For each message where the elapsed time is greater than the connection delivery timeout, the timed-out message is removed from the connection GMD area, a `GMD_FAILURE` message is created whose first field is the timed-out message, and TipcConnMsgProcess is called to asynchronously notify the sending process that there has been a delivery timeout. See TipcCbConnProcessGmdFailure for more information on processing `GMD_FAILURE` messages. Because TipcConnRead only checks for delivery timeouts after reading in messages, be careful to ensure that the timeout used with TipcConnRead is not larger than the connection delivery timeout property.

TipcConnRead then inserts the new messages according to priority into the connection's queue of available messages and calls the connection's queue callbacks.

TipcConnRead calls TipcConnCheck with timeout to check for network failures and to prevent the process from waiting longer than the desired timeout. TipcConnRead blocks if the socket function `recv` blocks (that is, no data is available on the socket). If the read fails (because `recv` returned 0 or -1), the connection's error callbacks are called to clean up, and TipcConnRead returns `FALSE`.

Caution TipcConnRead is a low-level function that is normally not used directly by developers; it is normally called only by TipcConnMsgNext and TipcConnMsgSearch.

If TipcConnGmdFileCreate fails to create a file-based GMD area, a warning similar to this is output with TutWarning:

```
WARNING: Could not create GMD file(s) with base name
<local_node_test>
for connection <client:local:_node:test>.
The option Unique_Subject must be set to a value other
than the default (_workstation1_8668).
Switching to memory-only GMD.
```

If this warning appears, the process is not configured properly to use file-based GMD. See TipcConnGmdFileCreate for more information on creating GMD areas.

Because TipcConnRead returns as soon as at least one message is read in, message priorities often have a limited impact on ordering. Large messages sent over a network are often fragmented by the network hardware and cause TipcConnRead to never read multiple large messages in one call. On systems like Solaris 2.X local sockets are not byte streams but are record-oriented devices. On these systems, if several small messages are flushed with TipcConnFlush one at a time by the sender, TipcConnRead does not read several messages at a time and sort them by priority.

See Also TipcConnCheck

Examples This example reads data that is immediately available on a connection:

```
if (!TipcConnRead(conn, 0.0)) {
    return; /* error */
}
```


TipcConnReadCbCreate

Name	TipcConnReadCbCreate — create a read callback in a connection
Synopsis	<pre>T_CB TipcConnReadCbCreate(<i>conn</i>, <i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_CONN <i>conn</i>; T_IPC_MT <i>mt</i>; T_IPC_CONN_READ_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>conn</i> — connection to create callback for <i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcConnReadCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcConnReadCbCreate creates a read callback in a connection. These callbacks are called when a new message is received from the connection's socket, which occurs in TipcConnRead. Connection read callbacks are useful for message file logging and preprocessing of messages before the normal processing with TipcConnMsgProcess. See the <i>TIBCO SmartSockets Utilities</i> for more information on callbacks. A read callback is usually created for a specific message type and connection. If the message type is null, then a global read callback is created for all message types on that connection.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnReadCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example creates a global connection read callback that writes all received messages to a message file:

```
void T_ENTRY my_conn_read_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_READ_CB_DATA data;
T_CB_ARG arg; /* really (T_IPC_MSG_FILE) */
{
    T_IPC_MSG msg = data->msg;
    T_IPC_MSG_FILE msg_file = arg;

    if (!TipcMsgFileWrite(msg_file, msg)) {
        return; /* error */
    }
} /* my_conn_read_cb */

/* ===== */
/*...code from calling function is below */

T_IPC_MSG_FILE msg_file;

msg_file = TipcMsgFileCreate("my_log.msg",
                             T_IPC_MSG_FILE_CREATE_WRITE);
if (msg_file == NULL) {
    return; /* error */
}

/* global connection read callback */
if (TipcConnReadCbCreate(conn, NULL, my_conn_read_cb, msg_file)
    == NULL) {
    return; /* error */
}
```

TipcConnReadCbLookup

Name	TipcConnReadCbLookup — look up a read callback in a connection
Synopsis	<pre>T_CB TipcConnReadCbLookup(<i>conn</i>, <i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_CONN <i>conn</i>; T_IPC_MT <i>mt</i>; T_IPC_CONN_READ_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>conn</i> — connection to look up callback for <i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcConnReadCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcConnReadCbLookup looks up a connection read callback with the desired message type, function, and argument in a connection. These callbacks are called when a new message is received from the connection's socket, which occurs in TipcConnRead. See TipcConnReadCbCreate for more information on connection read callbacks.
Caution	None
See Also	TipcConnReadCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example looks up a NUMERIC_DATA connection read callback and a global connection read callback:

```
mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

cb = TipcConnReadCbLookup(conn, mt, my_conn_read_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
/* global connection read callback */
cb = TipcConnReadCbLookup(conn, NULL, my_global_read_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
```

TipcConnSetAutoFlushSize

Name	TipcConnSetAutoFlushSize — set the auto flush size of a connection
Synopsis	<pre>T_BOOL TipcConnSetAutoFlushSize(conn, auto_flush_size) T_IPC_CONN conn; T_INT4 auto_flush_size;</pre>
Arguments	<i>conn</i> — connection to set auto flush size for <i>auto_flush_size</i> — auto flush size (in bytes)
Return Values	TRUE if the auto flush size was successfully set for the connection, FALSE otherwise.
Diagnostics	If TipcConnSetAutoFlushSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null • T_ERR_VAL_TOO_SMALL — <i>auto_flush_size</i> was negative
Description	TipcConnSetAutoFlushSize sets the auto flush size of a connection. The auto flush size is used to control how much outgoing data (in bytes) is buffered before being automatically flushed to the other end of connection. The default for the auto flush size is 8192 bytes. The auto flush size 0 can be used to disable all buffering and force each outgoing message to be flushed immediately. The auto flush size T_IPC_NO_AUTO_FLUSH_SIZE can be used to disable all automatic flushing of outgoing messages. See TipcConnGetAutoFlushSize for more information on auto flush sizes.
Caution	None
See Also	TipcConnGetAutoFlushSize
Examples	This example sets the auto flush size of a connection with TipcConnSetAutoFlushSize to 0 bytes, which causes all outgoing messages to be automatically flushed when sent: <pre>if (!TipcConnSetAutoFlushSize(conn, 0)) { return; /* error */ }</pre>

TipcConnSetBlockMode

Name TipcConnSetBlockMode — set the block mode of a connection

Synopsis

```
T_BOOL TipcConnSetBlockMode(conn, block_mode)
T_IPC_CONN conn;
T_BOOL block_mode;
```

Arguments *conn* — connection to set block mode for
block_mode — block mode

Return Values TRUE if block mode was successfully set for connection, FALSE otherwise.

Diagnostics If TipcConnSetBlockMode fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *conn* was null
- T_ERR_VAL_INVALID — *block_mode* was not TRUE or FALSE

Description TipcConnSetBlockMode sets the block mode of a connection. The connection block mode controls whether or not the process blocks waiting for read and write operations to complete. See TipcConnGetBlockMode for more information on connection block modes.

Caution None

See Also TipcConnGetBlockMode, TipcConnFlush, TipcConnRead

Examples This example sets the block mode of a connection to FALSE:

```
if (!TipcConnSetBlockMode(conn, FALSE)) {
    return; /* error */
}
```

TipcConnSetGmdMaxSize

Name	TipcConnSetGmdMaxSize — set the GMD area maximum size of a connection
Synopsis	<pre>T_BOOL TipcConnSetGmdMaxSize(conn, gmd_max_size) T_IPC_CONN conn; T_UINT4 gmd_max_size;</pre>
Arguments	<i>conn</i> — connection to set GMD area maximum size for <i>gmd_max_size</i> — connection GMD area maximum size (in bytes)
Return Values	TRUE if the GMD area maximum size was successfully set for the connection, FALSE otherwise.
Diagnostics	If TipcConnSetGmdMaxSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null • T_ERR_VAL_INVALID — the connection is not set up for GMD (TipcConnGmdFileCreate must be called before TipcConnSetGmdMaxSize)
Description	TipcConnSetGmdMaxSize sets the GMD area maximum size of a connection. A GMD area holds guaranteed message delivery information for both incoming and outgoing messages. There are two types of guaranteed message delivery: file-based GMD and memory-only GMD. See TipcConnGmdFileCreate for information on GMD areas. For file-based GMD this limit is the maximum file size, and for memory-only GMD this limit is the maximum amount of memory allowed. The default GMD area maximum size is 0, which means that no GMD area maximum size limit checking is performed. The connection GMD area maximum size and the connection delivery timeout can be used to constrain the amount of system resources used for GMD by space and by time. If the connection GMD area maximum size is exceeded, then no further messages can be sent with guaranteed delivery (TipcConnMsgSend returns FALSE and sets the global SmartSockets error number to T_ERR_VAL_TOO_LARGE) until some unacknowledged previously sent messages are acknowledged.
Caution	None
See Also	TipcConnGmdFileCreate, TipcConnGetGmdMaxSize, TipcSrvGetGmdMaxSize

Examples This example sets the GMD area maximum size of a connection to one million bytes:

```
if (!TipcConnSetGmdMaxSize(conn, 1000000)) {  
    return; /* error */  
}
```


TipcConnSetSocket

Name	TipcConnSetSocket — set the socket of a connection
Synopsis	<pre>T_BOOL TipcConnSetSocket(<i>conn</i>, <i>socket</i>) T_IPC_CONN <i>conn</i>; T_INT4 <i>socket</i>;</pre>
Arguments	<i>conn</i> — connection to set socket for <i>socket</i> — connection socket
Return Values	TRUE if socket was successfully set for connection, FALSE otherwise.
Diagnostics	If TipcConnSetSocket fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>conn</i> was null
Description	TipcConnSetSocket sets the socket file descriptor of a connection. A socket is an operating system device which provides a communication link to another process. All data sent to and received from the connection is transmitted on this socket.
Caution	The connection socket file descriptor should be set only by advanced applications.
See Also	TipcConnGetSocket
Examples	This example sets the socket of a connection: <pre>socket = my_create_override_socket(); if (!TipcConnSetSocket(conn, socket)) { return; /* error */ }</pre>

TipcConnSetTimeout

Name	TipcConnSetTimeout — set a timeout property of a connection
Synopsis	<pre>T_BOOL TipcConnSetTimeout(<i>conn</i>, <i>timeout</i>, <i>value</i>) T_IPC_CONN <i>conn</i>; T_IPC_TIMEOUT <i>timeout</i>; T_REAL8 <i>value</i>;</pre>
Arguments	<i>conn</i> — connection to set timeout for <i>timeout</i> — which timeout property to set <i>value</i> — timeout value (in seconds)
Return Values	TRUE if the timeout property was successfully set for the connection, FALSE otherwise.
Diagnostics	If TipcConnSetTimeout fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null • T_ERR_VAL_INVALID — <i>timeout</i> was not a valid TIMEOUT value • T_ERR_VAL_TOO_SMALL — <i>value</i> was negative
Description	TipcConnSetTimeout sets a timeout property of a connection. The timeout properties control how often (in seconds) the connection is checked for possible network failures. • If <i>timeout</i> is T_IPC_TIMEOUT_READ, TipcConnSetTimeout sets the read timeout of a connection. • If <i>timeout</i> is T_IPC_TIMEOUT_WRITE, TipcConnSetTimeout sets the write timeout of a connection. • If <i>timeout</i> is T_IPC_TIMEOUT_KEEP_ALIVE, TipcConnSetTimeout sets the keep alive timeout of a connection. • If <i>timeout</i> is T_IPC_TIMEOUT_DELIVERY, TipcConnSetTimeout sets the delivery timeout of a connection. See TipcConnGetTimeout for information on connection timeouts.
Caution	None
See Also	TipcConnCheck, TipcConnFlush, TipcConnGetTimeout, TipcConnRead

Examples This sets the read timeout of a connection with `TipcConnSetTimeout` to 10 seconds:

```
if (!TipcConnSetTimeout(conn, T_IPC_TIMEOUT_READ, 10.0)) {  
    return; /* error */  
}
```

TipcConnTrafficGetBytesRecv8

Name TipcConnTrafficGetBytesRecv — get the total number of bytes received on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

TipcConnTrafficGetBytesRecv — get the total number of bytes received on a connection

Synopsis

```
T_BOOL TipcConnTrafficGetBytesRecv8(conn, bytes_recv_return)
T_IPC_CONN conn;
T_INT8 *bytes_recv_return;
```

Arguments *conn* — connection to get information for
bytes_recv_return — returned number of bytes

Return Values TRUE if the number of bytes was sucessfully retrieved, FALSE otherwise.

Diagnostics If TipcConnTrafficGetBytesRecv8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *conn* or *bytes_recv_return* was null

Description TipcConnTrafficBetBytesRecv8 retrieves the total number of bytes received on a connection.

See Also TipcConnTrafficGetBytesSent8

Examples This example retrieves the number of bytes received on a connection:

```
T_IPC_CONN conn;
T_INT8 bytes_recv;
.
.
.
if (!TipcConnTrafficGetBytesRecv8(conn, &bytes_recv)) {
    return; /* error */
}
```

TipcConnTrafficGetBytesSent8

Name TipcConnTrafficGetBytesSent8 — get the total number of bytes sent on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supercedes it. We retain the older form only for backward source compatibility.

TipcConnTrafficGetBytesSent — get the total number of bytes sent on a connection

Synopsis `T_BOOL TipcConnTrafficGetBytesSent(conn, bytes_sent_return)`
 `T_IPC_CONN conn;`
 `T_INT8 *bytes_sent_return;`

Arguments *conn* — connection to get information for
 bytes_recv_return — returned number of bytes

Return Values TRUE if the number of bytes was successfully retrieved, FALSE otherwise.

Diagnostics If TipcConnTrafficGetBytesSent8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *conn* or *bytes_sent_return* was null

Description TipcConnTrafficBetBytesSent8 retrieves the total number of bytes received on a connection.

See Also TipcConnTrafficGetBytesRecv8

Examples This example retrieves the number of bytes sent on a connection:

```
T_IPC_CONN conn;
T_INT8 bytes_sent;
.
.
.
if (!TipcConnTrafficGetBytesSent8(conn, &bytes_sent)) {
    return; /* error */
}
```

TipcConnTrafficGetMsgsRecv8

Name

TipcConnTrafficGetMsgsRecv8 — get the total number of messages received on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

TipcConnTrafficGetMsgsRecv — get the total number of messages received on a connection

Synopsis

```
T_BOOL TipcConnTrafficGetMsgsRecv(conn, msgs_recv_return)
T_IPC_CONN conn;
T_INT8 *msgs_recv_return;
```

Arguments

conn — connection to get information for

msgs_recv_return — returned number of messages

Return Values

TRUE if the number of messages was sucessfully retrieved, FALSE otherwise.

Diagnostics

If TipcConnTrafficGetMsgsRecv8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *conn* or *msgs_recv_return* was null

Description

TipcConnTrafficGetMsgsRecv8 retrieves the total number of messages received on a connection.

See Also

TipcConnTrafficGetBytesSent8

Examples

This example retrieves the number of messages received on a connection:

```
T_IPC_CONN conn;
T_INT8 msgs_recv;
.
.
.
if (!TipcConnTrafficGetMsgsRecv8(conn, &msgs_recv)) {
    return; /* error */
}
```

TipcConnTrafficGetMsgsSent8

Name TipcConnTrafficGetMsgSent8 — get the total number of messages sent on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

TipcConnTrafficGetMsgSent — get the total number of messages sent on a connection

Synopsis `T_BOOL TipcConnTrafficGetMsgsSent8(conn, msgs_sent_return)`
 `T_IPC_CONN conn`
 `T_INT8 *msgs_sent_return;`

Arguments *conn* — connection to get information for
 msgs_sent_return — returned number of messages

Return Values TRUE if the number of messages was successfully retrieved, FALSE otherwise.

Diagnostics If TipcConnTrafficGetMsgsSent8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *conn* or *msgs_sent_return* was null

Description TipcConnTrafficGetMsgsSent8 retrieves the total number of messages sent on a connection.

See Also TipcConnTrafficGetBytesRecv8

Examples This example retrieves the number of messages sent on a connection:

```
T_IPC_CONN conn;
T_INT8 msgs_sent;
.
.
.
if (!TipcConnTrafficGetMsgsSent8(conn, &msgs_sent)) {
    return; /* error */
}
```

TipcConnUnlock

Name	TipcConnUnlock — release exclusive access to a connection
Synopsis	<pre>T_BOOL TipcConnUnlock(conn) T_IPC_CONN conn;</pre>
Arguments	<i>conn</i> — connection to unlock
Return Values	TRUE if connection was successfully unlocked, FALSE if error occurred.
Diagnostics	If TipcConnUnlock fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR— <i>conn</i> was null • any error number from TutMutexUnlock
Description	TipcConnUnlock unlocks all of the internal synchronization objects within the connection.
Caution	Every call to TipcConnLock must be balanced by a call to TipcConnUnlock.
See Also	TipcConnLock
Examples	This example locks and unlocks a connection: <pre>if (!TipcConnLock(conn)) { return; /* error */ }</pre> <i>/* At this point, several TipcConn* functions could be called and there would be no risk of any other thread accessing the connection. The connection must be unlocked with TipcConnUnlock when exclusive access is no longer needed. */</i> <pre>if (!TipcConnUnlock(conn)) { return; /* error */ }</pre>

TipcConnWriteCbCreate

Name	TipcConnWriteCbCreate — create a write callback in a connection
Synopsis	<pre>T_CB_TipcConnWriteCbCreate(conn, mt, func, arg) T_IPC_CONN conn; T_IPC_MT mt; T_IPC_CONN_WRITE_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>conn</i> — connection to create callback for <i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcConnWriteCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcConnWriteCbCreate creates a write callback in a connection. These callbacks are called when a message is buffered to be sent through a connection's socket, which occurs in TipcConnMsgSend. Connection write callbacks are useful for message file logging and processing of messages before they are actually sent. See <i>TIBCO SmartSockets Utilities</i> for more information on callbacks. A write callback is usually created for a specific message type and connection. If the message type is null, then a global write callback is created for all message types on that connection.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnWriteCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example creates a global connection write callback that writes all sent messages to a message file:

```
void T_ENTRY my_conn_write_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_WRITE_CB_DATA data;
T_CB_ARG arg; /* really (T_IPC_MSG_FILE) */
{
    T_IPC_MSG msg = data->msg;
    T_IPC_MSG_FILE msg_file = arg;

    if (!TipcMsgFileWrite(msg_file, msg)) {
        return; /* error */
    }
} /* my_conn_write_cb */

/* ===== */
/*...code from calling function is below */

msg_file = TipcMsgFileCreate("my_log.msg",
                             T_IPC_MSG_FILE_CREATE_WRITE);
if (msg_file == NULL) {
    return; /* error */
}

/* global connection write callback */
if (TipcConnWriteCbCreate(conn, NULL, my_conn_write_cb, msg_file)
    == NULL) {
    return; /* error */
}
```

TipcConnWriteCbLookup

Name	TipcConnWriteCbLookup — look up a write callback in a connection
Synopsis	<pre>T_CB TipcConnWriteCbLookup(<i>conn</i>, <i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_CONN <i>conn</i>; T_IPC_MT <i>mt</i>; T_IPC_CONN_WRITE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>conn</i> — connection to look up callback for <i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcConnWriteCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>conn</i> was null or <i>func</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcConnWriteCbLookup looks up a connection write callback with the desired message type, function, and argument in a connection. These callbacks are called when a message is buffered to be sent through a connection's socket, which occurs in TipcConnMsgSend. See TipcConnWriteCbCreate for information on connection write callbacks.
Caution	None
See Also	TipcConnWriteCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example looks up a NUMERIC_DATA connection write callback and a global connection write callback:

```
mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

cb = TipcConnWriteCbLookup(conn, mt, my_conn_write_cb, NULL);
if (cb == NULL) {
    return; /* error */
}

/* global connection write callback */
cb = TipcConnWriteCbLookup(conn, NULL, my_global_write_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
```

This chapter covers the API functions:

- *TipcDeliveryModeToStr*
- *TipcDispatcherCreate*
- *TipcDispatcherCreateDetached*
- *TipcDispatcherDestroy*
- *TipcDispatcherDispatch*
- *TipcDispatcherMainLoop*
- *TipcDispatcherSrvAdd*
- *TipcDispatcherSrvRemove*
- *TipcEventCreate*
- *TipcEventCreateConn*
- *TipcEventCreateMsg*
- *TipcEventCreateMsgType*
- *TipcEventCreateSocket*
- *TipcEventCreateTimer*
- *TipcEventDestroy*
- *TipcEventGetCheckMode*
- *TipcEventGetConn*
- *TipcEventGetData*
- *TipcEventGetDispatcher*
- *TipcEventGetInterval*
- *TipcEventGetSocket*
- *TipcEventGetType*
- *TipcEventSetInterval*
- *TipcFtToStr*
- *TipcGetGmdDir*

- *TipcInitCommands*
- *TipcInitThreads*
- *TipcLbModeToStr*

TipcDeliveryModeToStr

Name	TipcDeliveryModeToStr — convert a message delivery mode to a string
Synopsis	<pre>T_BOOL TipcDeliveryModeToStr(<i>delivery_mode</i>, <i>delivery_mode_str_return</i>) T_IPC_DELIVERY_MODE <i>delivery_mode</i>; T_STR *<i>delivery_mode_str_return</i>;</pre>
Arguments	<i>delivery_mode</i> — delivery mode to convert <i>delivery_mode_str_return</i> — storage for string
Return Values	TRUE if the delivery mode was successfully converted to a string, FALSE otherwise.
Diagnostics	If TipcDeliveryModeToStr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>delivery_mode</i> was not a valid T_IPC_DELIVERY_MODE value • T_ERR_NULL_PTR — <i>delivery_mode_str_return</i> was null
Description	TipcDeliveryModeToStr converts a message delivery mode (such as T_IPC_DELIVERY_ALL) to a lower-case string (such as all).
Caution	If TipcDeliveryModeToStr returns FALSE, it does not store a value in <i>delivery_mode_str_return</i>. The value stored in <i>delivery_mode_str_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcStrToDeliveryMode
Examples	This example prints the delivery mode of a message: <pre>T_IPC_DELIVERY_MODE <i>delivery_mode</i>; T_STR <i>delivery_mode_str</i>; if (!TipcMsgGetDeliveryMode(msg, &<i>delivery_mode</i>)) { return; /* error */ } if (!TipcDeliveryModeToStr(<i>delivery_mode</i>, &<i>delivery_mode_str</i>)) { return; /* error */ } TutOut("Message has delivery mode %s (%d).\n", <i>delivery_mode_str</i>, <i>delivery_mode</i>);</pre>

TipcDispatcherCreate

Name	TipcDispatcherCreate — create a dispatcher
Synopsis	T_IPC_DISPATCHER TipcDispatcherCreate()
Arguments	None
Return Values	A new dispatcher if successful, NULL otherwise.
Diagnostics	None
Description	TipcDispatcherCreate creates a new dispatcher to which events can be registered and from which messages can be received on RTclient/RTserver connections.
Caution	A dispatcher must continuously manage incoming messages and events by calling TipcDispatcherDispatch or TipcDispatcherMainLoop. A dispatcher created with TipcDispatcherCreate which does not do this can create a situation that leads to unpredictable behavior in your application. You must use TipcDispatcherCreate if your SmartSockets application is a single thread, in which case you must ensure the calls to TipcDispatcherDispatch or TipcDispatcherMainLoop take place.
See Also	TipcDispatcherCreateDetached, TipcDispatcherDestroy
Examples	This example creates a dispatcher: <pre>T_IPC_DISPATCHER dispatcher; dispatcher = TipcDisaptcherCreate();</pre>

TipcDispatcherCreateDetached

Name	TipcDispatcherCreateDetached — create a dispatcher in a detached thread
Synopsis	<code>T_IPC_DISPATCHER TipcDispatcherCreateDetached()</code>
Arguments	None
Return Values	A new dispatcher if successful, NULL otherwise.
Diagnostics	None
Description	TipcDispatcherCreateDetached creates a new dispatcher to which events can be registered and from which messages can be received on RTclient/RTserver connections. A detached dispatcher runs continually in a background thread and does not stop until TipcDispatcherDestroy is called.
Caution	Before you use any thread function, you must call TipcInitThreads to initialize SmartSockets thread synchronization.
See Also	TipcDispatcherCreate, TipcDispatcherDestroy
Examples	This example creates a detached dispatcher: <pre>T_IPC_DISPATCHER dispatcher; dispatcher = TipcDisaptcherCreateDetached();</pre>

TipcDispatcherDestroy

Name	TipcDispatcherDestroy — destroy a dispatcher
Synopsis	<pre>T_BOOL TipcDispatcherDestroy(dispatcher) T_IPC_DISPATCHER dispatcher;</pre>
Arguments	<i>dispatcher</i> — the dispatcher to destroy
Return Values	TRUE if the dispatcher was successfully destroyed, FALSE otherwise.
Diagnostics	If TipcDispatcherDestroy fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>dispatcher</i> was null • T_ERR_UNSUPPORTED — the current thread is not the owner of the dispatcher
Description	TipcDispatcherDestroy frees resources associated with a dispatcher.
Caution	This function can only be called from the same thread that created the dispatcher.
See Also	TipcDispatcherCreate, TipcDispatcherCreateDetached
Examples	This example creates and destroys a dispatcher: <pre>T_IPC_DISPATCHER dispatcher; dispatcher = TipcDispatcherCreate(); /* do work here */ if (!TipcDispatcherDestroy(dispatcher)) { return; /* error */ }</pre>

TipcDispatcherDispatch

Name	TipcDispatcherDispatch — dispatch triggered events and exit
Synopsis	<pre>T_BOOL TipcDispatcherDispatch(dispatcher, timeout) T_IPC_DISPATCHER dispatcher; T_REAL8 timeout;</pre>
Arguments	<i>dispatcher</i> — the dispatcher <i>timeout</i> — the amount of time to wait for triggered events
Return Values	TRUE if the dispatcher successfully dispatched events, FALSE otherwise.
Diagnostics	If TipcDispatcherDispatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>dispatcher</i> was null • T_ERR_VAL_INVALID — <i>timeout</i> was negative • T_ERR_TIMEOUT_REACHED — no events were dispatched • T_ERR_UNSUPPORTED — the current thread is not the owner of the dispatcher
Description	TipcDispatcherDispatch immediately invokes any triggered events and then exits event if <i>timeout</i> seconds has not elapsed. If there are no triggered events, it waits for up to <i>timeout</i> seconds for events to be triggered.
Caution	This function can only be called from the same thread that created the dispatcher.
See Also	TipcDispatcherMainLoop
Examples	This example dispatches events: <pre>if (!TipcDispatcherDispatch(dispatcher, 1.0)) { return; /* error */ }</pre>

TipcDispatcherMainLoop

Name	TipcDispatcherMainLoop — dispatch events for a specified amount of time
Synopsis	<pre>T_BOOL TipcDispatcherMainLoop(dispatcher, timeout) T_IPC_DISPATCHER dispatcher; T_REAL8 timeout;</pre>
Arguments	<i>dispatcher</i> — the dispatcher <i>timeout</i> — the amount of time in which to manage events
Return Values	TRUE if the dispatcher successfully dispatched events, FALSE otherwise.
Diagnostics	If TipcDispatcherMainLoop fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>dispatcher</i> was null • T_ERR_VAL_INVALID — <i>timeout</i> was negative • T_ERR_UNSUPPORTED — the current thread is not the owner of the dispatcher
Description	TipcDispatcherMainLoop manages events for <i>timeout</i> seconds.
Caution	This function can only be called from the same thread that created the dispatcher.
See Also	TipcDispatcherDispatch
Examples	This example dispatches events: <pre>if (!TipcDispatcherMainLoop(dispatcher, T_TIMEOUT_FOREVER)) { return; /* error */ }</pre>

TipcDispatcherSrvAdd

Name	TipcDispatcherSrvAdd — add an RTserver connection to a dispatcher
Synopsis	<pre>T_BOOL TipcDispatcherSrvAdd(dispatcher, srv) T_IPC_DISPATCHER dispatcher; T_IPC_SRV srv;</pre>
Arguments	<i>dispatcher</i> — the dispatcher <i>srv</i> — connection to the RTserver
Return Values	TRUE if the RTserver connection was successfully added to a dispatcher, FALSE otherwise.
Diagnostics	If TipcDispatcherSrvAdd fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>srv</i> or <i>dispatcher</i> was null • T_ERR_ALREADY_EXISTS — <i>srv</i> was already added to <i>dispatcher</i>. You misspelled the identifier or tried to add the connection twice.
Description	TipcDispatcherSrvAdd adds an RTserver connection to a dispatcher. A dispatcher reads messages from multiple connections at one time and is an alternative to calling TipcSrvConnMainLoop, or TipcSrvConnMsgNext, TipcSrvConnMsgProcess and TipcMsgDestroy on each connection individually.
Caution	Calling TipcSrvConnMsgNext, TipcSrvConnMsgSearch, TipcSrvConnMsgSearchType, TipcSrvConnMsgSendRpc or TipcSrvConnMainLoop might cause a dispatcher to block until the called function returns.
See Also	TipcDispatcherSrvRemove, TipcDispatcherDispatch, TipcDispatcherMainLoop
Examples	This example adds an RTserver connection to a dispatcher: <pre>if (!TipcDispatcherSrvAdd(dispatcher, srv)) { return; /* error */ }</pre>

TipcDispatcherSrvRemove

Name	TipcDispatcherSrvRemove — remove an RTserver connection from a dispatcher
Synopsis	<pre>T_BOOL TipcDispatcherSrvRemove(dispatcher, srv) T_IPC_DISPATCHER dispatcher; T_IPC_SRV srv;</pre>
Arguments	<i>dispatcher</i> — the dispatcher <i>srv</i> — connection to the RTserver
Return Values	TRUE if the RTserver connection was successfully removed from a dispatcher, FALSE otherwise.
Diagnostics	If TipcDispatcherSrvRemove fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>srv</i> or <i>dispatcher</i> was null • T_ERR_DOESNT_EXIST — there is no <i>srv</i> for <i>dispatcher</i>. You misspelled the identifier or tried to remove the connection more than one time.
Description	TipcDispatcherSrvRemove removes an RTserver connection from a dispatcher.
Caution	None
See Also	TipcDispatcherSrvAdd, TipcDispatcherDispatch, TipcDispatcherMainLoop
Examples	This example removes an RTserver connection from a dispatcher: <pre>if (!TipcDispatcherSrvRemove(dispatcher, srv)) { return; /* error */ }</pre>

TipcEventCreate

Name	TipcEventCreate — create a user event
Synopsis	<pre>T_IPC_EVENT TipcEventCreate(dispatcher, data, event_func, event_arg) T_IPC_DISPATCHER dispatcher; T_PTR data; T_IPC_EVENT_FUNC event_func; T_PTR event_arg;</pre>
Arguments	<i>dispatcher</i> — the dispatcher on which to invoke the event <i>data</i> — data defined by the user that is associated with the event <i>event_func</i> — the function to be invoked <i>event_arg</i> — an optional argument to pass to the <i>event_func</i>
Return Values	A new event if successful, NULL otherwise.
Diagnostics	If TipcEventCreate fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>dispatcher</i>, <i>data</i> or <i>event_func</i> was null
Description	TipcEventCreate creates an event of type T_IPC_EVENT_USER to be invoked immediately on the specified dispatcher. The <i>data</i> can be any arbitrary user specified data. These events are useful for inter-thread communication.
Caution	None
See Also	TipcEventGetType, TipcEventGetData
Examples	This example creates a user event used to destroy and clean up a socket event: <pre>void T_ENTRY eventDestroyFunc(T_IPC_EVENT event, T_IPC_EVENT_DATA data, T_PTR arg) { T_IPC_EVENT event_to_destroy; T_INT4 socket = (T_INT4)arg; if (!TipcEventGetData(event, &event_to_destroy)) { return; /* error */ } /* Destroy the socket event */ if (!TipcEventDestroy(event_to_destroy)) { return; /* error */ } }</pre>

```

    /* Close the socket */
    close(socket);

}

/* ===== */
/*...code from main program is below */

socket_event = TipcEventCreateSocket(dispatcher, socket,
                                     T_IO_CHECK_READ,
                                     readFunc, T_NULL);

if (socket_event == NULL) {
    return; /* error */
}

/*
 * This is the proper way to destroy an event in a dispatcher
 * running in another thread, such as a detached dispatcher.
 */
if (NULL == TipcEventCreate(dispatcher, (T_PTR)socket_event,
                             eventDestroyFunc,
                             (T_PTR)socket);

    return; /* error */
}

```


TipcEventCreateConn

Name	TipcEventCreateConn — create a connection event
Synopsis	<pre>T_IPC_EVENT TipcEventCreatConn(dispatcher, conn, check_mode, event_func, event_arg) T_IPC_DISPATCHER dispatcher; T_IPC_CONN conn; T_IO_CHECK_MODE check_mode; T_IPC_EVENT_FUNC event_func; T_PTR event_arg;</pre>
Arguments	<i>dispatcher</i> — the dispatcher on which to invoke the connection event <i>conn</i> — the connection to wait for availability on <i>check_mode</i> — the kind of availability to wait on: T_IO_CHECK_READ or T_IO_CHECK_WRITE <i>event_func</i> — the function to be invoked <i>event_arg</i> — an optional argument to pass to the <i>event_func</i>
Return Values	A new event if successful, NULL otherwise.
Diagnostics	If TipcEventCreateConn fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>dispatcher</i>, <i>conn</i> or <i>event_func</i> was null • T_ERR_VAL_INVALID — <i>check_mode</i> was invalid
Description	TipcEventCreateConn creates a connection event of type T_IPC_EVENT_CONN to be invoked when <i>conn</i> is available for reading or writing, depending on the value of <i>check_mode</i> .
Caution	When a connection event is created or destroyed from a thread other than the thread dispatching the events, a user event is added to the dispatcher to perform the actual creation or destruction. To ensure that event data, such as a socket or a connection, is not destroyed before the event is removed from the dispatcher, create a user event to destroy the event and to perform any necessary cleanup. See the example for TipcEventCreate.
See Also	TipcEventGetType, TipcEventGetCheckMode, TipcEventGetConn

Examples This example creates a connection event:

```
void T_ENTRY connFunc(T_IPC_EVENT event, T_IPC_EVENT_DATA data,
                     T_PTR arg)
{
    T_IO_CHECK_MODE check_mode;
    T_IPC_CONN conn;

    if (!TipcEventGetCheckMode(event, &check_mode)) {
        return; /* error */
    }

    if (!TipcEventGetConn(event, &conn)) {
        return; /* error */
    }

    if (data->check_mode == T_IO_CHECK_READ) {
        TipcConnMainLoop(data->conn, 0.25);
    }
    else if (data->check_mode == T_IO_CHECK_WRITE) {
        TipcConnFlush(data->conn);
    }
}

/* ===== */
/*...code from main program is below */

read_event = TipcEventCreateConn(dispatcher, conn,
                                connFunc, T_NULL);
if (read_event == NULL) {
    return; /* error */
}

write_event = TipcEventCreateConn(dispatcher, socket,
                                connFunc, T_NULL);
if (write_event == NULL) {
    return; /* error */
}
```

TipcEventCreateMsg

Name	TipcEventCreateMsg — create a message event that is invoked based on a message's subject
Synopsis	<pre>T_IPC_EVENT TipcEventCreatMsg(dispatcher, srv, subject, event_func, event_arg) T_IPC_DISPATCHER dispatcher; T_IPC_SRV srv; T_STR subject; T_IPC_EVENT_FUNC event_func; T_PTR event_arg;</pre>
Arguments	<i>dispatcher</i> — the dispatcher on which to invoke the event <i>srv</i> — an RTserver connection to wait for messages on <i>subject</i> — the subject that triggers the event <i>event_func</i> — the function to be invoked <i>event_arg</i> — an optional argument to pass to the <i>event_func</i>
Return Values	A new event if successful, NULL otherwise.
Diagnostics	If TipcEventCreateMsg fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>dispatcher</i>, <i>srv</i> or <i>event_func</i> was null • T_ERR_VAL_INVALID — <i>subject</i> was invalid
Description	TipcEventCreateMsg creates a message event of type T_IPC_EVENT_MSG to be invoked when a message arrives on the <i>srv</i> connection whose destination matches <i>subject</i> .
Caution	The message must not be destroyed from within the event's callback function. SmartSockets manages the message and automatically destroys the message once all events are dispatched. If you need to keep a copy of the message, call TipcMsgClone.
See Also	TipcEventGetType, TipcEventCreateMsgType

Examples This example creates a message event that is triggered by a message's subject:

```
void T_ENTRY msgFunc(T_IPC_EVENT event, T_IPC_EVENT_DATA data,
                    T_PTR arg)
{
    TipcMsgPrint(data->msg, TutOut);
}

/* ===== */
/*...code from main program is below */

msg_event = TipcEventCreateMsg(dispatcher, srv, "/stocks",
msgFunc,
                                T_NULL);

if (msg_event == NULL) {
    return; /* error */
}
```

TipcEventCreateMsgType

Name	TipcEventCreateMsgType — create a message event that is invoked based on a message's type
Synopsis	<pre>T_IPC_EVENT TipcEventCreatMsgType(dispatcher, srv, mt, event_func, event_arg) T_IPC_DISPATCHER dispatcher; T_IPC_SRV srv; T_IPC_MT mt; T_IPC_EVENT_FUNC event_func; T_PTR event_arg;</pre>
Arguments	<i>dispatcher</i> — the dispatcher on which to invoke the event <i>srv</i> — an RTserver connection to wait for messages on <i>mt</i> — the message type that triggers the event <i>event_func</i> — the function to be invoked <i>event_arg</i> — an optional argument to pass to the <i>event_func</i>
Return Values	A new event if successful, NULL otherwise.
Diagnostics	If TipcEventCreateMsgType fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>dispatcher</i>, <i>srv</i>, <i>mt</i> or <i>event_func</i> was null
Description	TipcEventCreateMsgType creates a message event of type T_IPC_EVENT_MSG_TYPE to be invoked when a message arrives on the <i>srv</i> connection whose destination matches the <i>mt</i> .
Caution	The message must not be destroyed from within the event's callback function. SmartSockets manages the message and automatically destroys the message once all events are dispatched. If you need to keep a copy of the message, call TipcMsgClone.
See Also	TipcEventGetType, TipcEventCreateMsg

Examples This example creates a message event triggered by a message's type:

```
void T_ENTRY msgTypeFunc(T_IPC_EVENT event, T_IPC_EVENT_DATA data,
                        T_PTR arg)
{
    TipcMsgPrint(data->msg, TutOut);
}

/* ===== */
/*...code from main program is below */

msg_event = TipcEventCreateMsgType(dispatcher, srv, mt,
                                    msgTypeFunc, T_NULL);
if (msg_event == NULL) {
    return; /* error */
}
```

TipcEventCreateSocket

Name	TipcEventCreateSocket — create a socket event
Synopsis	<pre>T_IPC_EVENT TipcEventCreatSocket(dispatcher, socket, check_mode, event_func, event_arg) T_IPC_DISPATCHER dispatcher; T_INT4 socket; T_IO_CHECK_MODE check_mode; T_IPC_EVENT_FUNC event_func; T_PTR event_arg;</pre>
Arguments	<i>dispatcher</i> — the dispatcher on which to invoke the event <i>socket</i> — the socket to wait for availability on <i>check_mode</i> — the type of availability to wait on: T_IO_CHECK_READ or T_IO_CHECK_WRITE <i>event_func</i> — the function to be invoked <i>event_arg</i> — an optional argument to pass to the <i>event_func</i>
Return Values	A new event if successful, NULL otherwise.
Diagnostics	If TipcEventCreateSocket fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>dispatcher</i> or <i>event_func</i> was null • T_ERR_VAL_INVALID — <i>socket</i> or <i>check_mode</i> was invalid
Description	TipcEventCreateSocket creates a socket event of type T_IPC_EVENT_SOCKET to be invoked when <i>socket</i> is available for reading or writing, depending on the value of <i>check_mode</i> .
Caution	When an event is created or destroyed from a thread other than the thread dispatching the events, a user event is added to the dispatcher to perform the actual creation or destruction. To ensure that event data, such as a socket or a connection, is not destroyed before the event is removed from the dispatcher, create a user event to destroy the event and to perform any necessary cleanup. See the example for TipcEventCreate.
See Also	TipcEventGetType, TipcEventGetCheckMode, TipcEventGetSocket

Examples This example creates a socket event:

```
void T_ENTRY socketFunc(T_IPC_EVENT event, T_IPC_EVENT_DATA data,
                       T_PTR arg)
{
    T_IO_CHECK_MODE check_mode;
    T_INT4 socket;

    if (!TipcEventGetCheckMode(event, &check_mode)) {
        return; /* error */
    }

    if (!TipcEventGetSocket(event, &socket)) {
        return; /* error */
    }

    if (check_mode == T_IO_CHECK_READ) {
        read(socket);
    }
    else if (check_mode == T_IO_CHECK_WRITE) {
        write(socket);
    }
}

/* ===== */
/*...code from main program is below */

read_event = TipcEventCreateSocket(dispatcher, socket,
                                   T_IO_CHECK_READ,
                                   socketFunc, T_NULL);

if (read_event == NULL) {
    return; /* error */
}

write_event = TipcEventCreateSocket(dispatcher, socket,
                                   T_IO_CHECK_WRITE, socketFunc,
                                   T_NULL);
if (write_event == NULL) {
    return; /* error */
}
```


TipcEventCreateTimer

Name	TipcEventCreateTimer — create a timer event
Synopsis	<pre>T_IPC_EVENT TipcEventCreatTimer(dispatcher, interval, event_func, event_arg) T_IPC_DISPATCHER dispatcher; T_REAL8 interval; T_IPC_EVENT_FUNC event_func; T_PTR event_arg;</pre>
Arguments	<i>dispatcher</i> — the dispatcher on which to invoke the event <i>interval</i> — the timer interval <i>event_func</i> — the function to be invoked <i>event_arg</i> — an optional argument to pass to the <i>event_func</i>
Return Values	A new event if successful, NULL otherwise.
Diagnostics	If TipcEventCreateTimer fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>dispatcher</i> or <i>event_func</i> was null • T_ERR_VAL_INVALID — <i>interval</i> is negative
Description	TipcEventCreateTimer creates a timer event of type T_IPC_EVENT_TIMER to be invoked every <i>interval</i> seconds on the specified dispatcher. If the timer is to be invoked only once, destroy the event in the <i>event_func</i> after the first time it executes.
Caution	None
See Also	TipcEventGetType, TipcEventGetInterval, TipcEventSetInterval

Examples This example creates a timer event and modifies the event's interval within the invoked function:

```
void T_ENTRY timerFunc(T_IPC_EVENT event, T_IPC_EVENT_DATA data,
                      T_PTR arg)
{
    T_REAL8 interval;

    if (!TipcEventGetInterval(event, &interval)) {
        return; /* error */
    }

    /* Double the interval */
    if (!TipcEventSetInterval(event, interval * 2.0)) {
        return; /* error */
    }
}

/* ===== */
/*...code from main program is below */

event = TipcEventCreateTimer(dispatcher, 1.0, timerFunc, T_NULL);
if (event == NULL) {
    return; /* error */
}
```

TipcEventDestroy

Name	TipcEventDestroy — destroy an event
Synopsis	<pre>T_BOOL TipcEventDestroy(event) T_IPC_EVENT event;</pre>
Arguments	<i>event</i> — the event to destroy
Return Values	TRUE if the event was successfully destroyed, FALSE otherwise.
Diagnostics	If TipcEventDestroy fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>event</i> was null
Description	TipcEventDestroy destroys the event.
Caution	When an event is created or destroyed from a thread other than the thread dispatching the events, a user event is added to the dispatcher to perform the actual creation or destruction. To ensure that event data, such as a socket or a connection, is not destroyed before the event is removed from the dispatcher, create a user event to destroy the event and to perform any necessary cleanup. See the example for TipcEventCreate.
See Also	TipcEventCreate
Examples	This example destroys an event: <pre>if (!TipcEventDestroy(event)) { return; /* error */ }</pre>

TipcEventGetCheckMode

Name	TipcEventGetCheckMode — get check mode value of a connection or socket event
Synopsis	<pre>T_BOOL TipcEventGetCheckMode(event, check_mode_return) T_IPC_EVENT event; T_IO_CHECK_MODE *check_mode_return;</pre>
Arguments	<i>event</i> — a connection or socket event <i>check_mode_return</i> — returned check mode
Return Values	TRUE if the check mode was successfully retrieved for the event, FALSE otherwise.
Diagnostics	If TipcEventGetCheckMode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>event</i> or <i>check_mode_return</i> was null • T_ERR_VAL_INVALID — <i>event</i> is not of type T_IPC_EVENT_SOCKET or T_IPC_EVENT_CONN
Description	TipcEventGetCheckMode returns the check mode value, T_IO_CHECK_READ or T_IO_CHECK_WRITE, of the event.
Caution	None
See Also	TipcEventCreateSocket, TipcEventCreateConn
Examples	This example returns the value of an event’s check mode: <pre>T_IO_CHECK_MODE check_mode; if (!TipcEventGetCheckMode(event, &check_mode)) { return; /* error */ }</pre>

TipcEventGetConn

Name	TipcEventGetConn — get an event’s connection information
Synopsis	<pre>T_BOOL TipcEventGetConn(event, conn_return) T_IPC_EVENT event; T_IPC_CONN *conn_return;</pre>
Arguments	<i>event</i> — a connection event <i>conn_return</i> — returned connection information
Return Values	TRUE if the connection was successfully retrieved for the event, FALSE otherwise.
Diagnostics	If TipcEventGetConn fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>event</i> or <i>connection_return</i> was null • T_ERR_VAL_INVALID — <i>event</i> is not of type T_IPC_EVENT_CONN
Description	TipcEventGetConn gets the event’s connection.
Caution	None
See Also	TipcEventCreateConn
Examples	This example returns the connection information for a connection event: <pre>T_IPC_CONN conn; if (!TipcEventGetConn(event, &conn)) { return; /* error */ }</pre>

TipcEventData

Name TipcEventData — get a user event’s data

Synopsis `T_BOOL TipcEventData(event, data_return)`
`T_IPC_EVENT event;`
`T_PTR *data_return;`

Arguments *event* — a user event
data_return — returned data

Return Values TRUE if the data was successfully retrieved for the event, FALSE otherwise.

Diagnostics If TipcEventData fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *event* or *data_return* was null
- T_ERR_VAL_INVALID — *event* is not of type T_IPC_EVENT_USER

Description TipcEventData gets the user-defined data for a user event.

Caution None

See Also TipcEventCreate

Examples This example returns the user-defined data for a user event:

```
T_PTR data;

if (!TipcEventData(event, &data)) {
    return; /* error */
}
```

TipcEventGetDispatcher

Name	TipcEventGetDispatcher — get the identifier of an event’s dispatcher
Synopsis	<pre>T_BOOL TipcEventGetDispatcher(event, dispatcher_return) T_IPC_EVENT event; T_IPC_DISPATCHER *dispatcher_return;</pre>
Arguments	<i>event</i> — an event (connection, message, socket, timer, or user event) <i>dispatcher_return</i> — returned dispatcher identifier
Return Values	TRUE if the dispatcher was successfully retrieved for the event, FALSE otherwise.
Diagnostics	If TipcEventGetDispatcher fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>event</i> or <i>dispatcher_return</i> was null
Description	TipcEventGetDispatcher returns the identifier of an event’s dispatcher.
Caution	None
See Also	TipcEventCreate
Examples	This example returns the identifier of an event’s dispatcher: <pre>T_IPC_DISPATCHER dispatcher; if (!TipcEventGetDispatcher(event, &dispatcher)) { return; /* error */ }</pre>

TipcEventGetInterval

Name TipcEventGetInterval — get a timer event’s time interval

Synopsis `T_BOOL TipcEventGetInterval(event, interval_return)`
`T_IPC_EVENT event;`
`T_REAL8 *interval_return;`

Arguments *event* — a timer event
interval_return — returned interval, in seconds

Return Values TRUE if the interval was successfully retrieved for the event, FALSE otherwise.

Diagnostics If TipcEventGetInterval fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *event* or *interval_return* was null
- T_ERR_VAL_INVALID — *event* is not of type T_IPC_EVENT_TIMER

Description TipcEventGetInterval returns the time interval of a timer event.

Caution None

See Also TipcEventCreateTimer, TipcEventSetInterval

Examples This example returns the time interval of a timer event.

```
T_REAL8 interval;  
  
if (!TipcEventGetInterval(event, &interval)) {  
    return; /* error */  
}
```


TipcEventGetSocket

Name	TipcEventGetSocket — get the socket information for a socket event
Synopsis	<pre>T_BOOL TipcEventGetSocket(<i>event</i>, <i>socket_return</i>) T_IPC_EVENT <i>event</i>; T_INT4 *<i>socket_return</i>;</pre>
Arguments	<i>event</i> — a socket event <i>socket_return</i> — returned socket
Return Values	TRUE if the socket was successfully retrieved for the event, FALSE otherwise.
Diagnostics	If TipcEventGetSocket fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>event</i> or <i>socket_return</i> was null • T_ERR_VAL_INVALID — <i>event</i> is not of type T_IPC_EVENT_SOCKET
Description	TipcEventGetSocket returns the socket information for a socket event.
Caution	None
See Also	TipcEventCreateSocket
Examples	This example returns the socket information for a socket event. <pre>T_INT4 socket; if (!TipcEventGetSocket(<i>event</i>, &socket)) { return; /* error */ }</pre>

TipcEventGetType

Name	TipcEventGetType — get the type of an event
Synopsis	<pre>T_BOOL TipcEventGetType(event, type_return) T_IPC_EVENT event; T_IPC_EVENT_TYPE *type_return;</pre>
Arguments	<i>event</i> — the event <i>type_return</i> — returned type
Return Values	TRUE if the type was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcEventGetType fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>event</i> or <i>type_return</i> was null
Description	TipcEventGetType returns an event type: • T_IPC_EVENT_CONN for a connection event • T_IPC_EVENT_MSG for a message event that is invoked depending on a message’s type • T_IPC_EVENT_MSG_TYPE for a message event that is invoked depending on a message’s subject • T_IPC_EVENT_SOCKET for a socket event • T_IPC_EVENT_TIMER for timer event • T_IPC_EVENT_USER for a user event
Caution	None
See Also	TipcEventCreate
Examples	This example returns an event’s type: <pre>T_IPC_EVENT_TYPE type; if (!TipcEventGetType(event, &type)) { return; /* error */ }</pre>

TipcEventSetInterval

Name	TipcEventSetInterval — set a timer event's interval
Synopsis	<pre>T_BOOL TipcEventSetInterval(event, interval) T_IPC_EVENT event; T_REAL8 interval;</pre>
Arguments	<i>event</i> — a timer event <i>interval</i> — new interval, in seconds
Return Values	TRUE if the interval was successfully set for the event, FALSE otherwise.
Diagnostics	If TipcEventSetInterval fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>event</i> was null • T_ERR_VAL_INVALID — <i>event</i> is not of type T_IPC_EVENT_TIMER or <i>interval</i> was negative
Description	TipcEventSetInterval sets an amount of time, in seconds, to wait before invoking a timer event.
Caution	None
See Also	TipcEventCreateTimer, TipcEventGetInterval
Examples	This example sets a time event's interval to 30 seconds: <pre>if (!TipcEventSetInterval(event, 30.0) { return; /* error */ }</pre>

TipcFtToStr

Name	TipcFtToStr — convert a field type to a string
Synopsis	<pre>T_BOOL TipcFtToStr(<i>ft</i>, <i>ft_str_return</i>) T_IPC_FT <i>ft</i>; T_STR *<i>ft_str_return</i>;</pre>
Arguments	<i>ft</i> — field type to convert <i>ft_str_return</i> — storage for string
Return Values	TRUE if the field type was successfully converted to a string, FALSE otherwise.
Diagnostics	If TipcFtToStr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>ft_str_return</i> was null • T_ERR_FT_INVALID — <i>ft</i> was not a valid T_IPC_FT
Description	TipcFtToStr converts a field type, such as T_IPC_FT_REAL8, to a lower-case string, such as REAL8.
Caution	If TipcFtToStr returns FALSE, it does not store a value in <i>ft_str_return</i>. The value stored in <i>ft_str_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcStrToFt

Examples This example converts all the field types in a message to strings and prints the strings out:

```
T_PTR print_one_field(msg, trav, arg)
T_IPC_MSG msg;
T_IPC_MSG_TRAV trav;
T_PTR arg; /* really (T_INT4 *) */
{
    T_STR str;
    T_IPC_FT type = (T_IPC_FT)*(trav->type_ptr);
    T_INT4 *counter_ptr = (T_INT4 *)arg;

    (*counter_ptr)++;
    if (!TipcFtToStr(type, &str)) {
        return; /* error */
    }
    TutOut("Field %d has type %s (%d).\n",
           *counter_ptr, str, type);

    return NULL; /* continue traversal */
} /* print_one_field */

void print_one_message(msg)
T_IPC_MSG msg;
{
    T_INT4 counter = 0;
    TipcMsgTraverse(msg, print_one_field, &counter);
} /* print_one_message */
```

TipcGetGmdDir

Name	TipcGetGmdDir — get name of directory where files are written for guaranteed message delivery
Synopsis	T_STR TipcGetGmdDir()
Arguments	None
Return Values	String containing directory name if successful, NULL otherwise.
Diagnostics	If TipcGetGmdDir fails, it returns NULL and sets the global SmartSockets error number to one of: - T_ERR_VAL_REQUIRED — the option Unique_Subject was not set to a value other than the default <i>_Node_Pid</i> - T_ERR_C — a C error has occurred. Call the function TutErrNumGetC to obtain the related C-specific error value.
Description	TipcGetGmdDir returns the name of the directory where guaranteed message delivery (GMD) files are kept. These messages are saved in files so that they can be resent after a program crashes and is restarted. The name of this directory is built from the value of the option Ipc_Gmd_Directory plus gmd plus the value of the option Unique_Subject. The generated components of the directory name are shrunk to 8 characters if they are too long for the native filesystem. (Windows Version 3.1 only allows 8-character file base names, and OpenVMS only allows 39-character file base names.) For example, a server process with the unique subject <i>server_us</i> communicating with a client process with the unique subject <i>client_us</i> using the logical connection name <i>tcp:workstation1:2042</i> would use the full pathname:

UNIX:
/tmp/rtworks/gmd/server_us/tcp_workstation1_2042_client_us.hsn

OpenVMS:
dka300:[ssuser.gmd.server_us]tcp_workstation1_2042_client_us.hsn

Windows:
c:\temp\rtworks\gmd\server_us\tcp_workstation1_2042_client_us.hsn

For best performance, this directory should be on a local file system.

Each process needs a unique directory where it can store its GMD files, and thus the value of the option `Unique_Subject` is used to build the name of this directory. Using files allows a program to be restarted and continue sending and receiving messages from exactly where it left off previously. The default value for the `Unique_Subject` option has the integer process identifier (PID) in it. Because the PID changes on UNIX and Windows systems when a program is rerun, to use GMD files the option `Unique_Subject` must be set to value other than the default `_Node_Pid`. If `Unique_Subject` is not explicitly set by the user, then file-based GMD cannot be used. See the page for `TipcSrvConnGmdFileCreate` for more information on GMD areas.

Caution The string returned by `TipcGetGmdDir` points into an internal data structure and should not be modified.

The most common cause of `TipcGetGmdDir` failing is not setting the option `Unique_Subject` to a value other than the default `_Node_Pid`. Each process that uses file-based GMD must properly configure `Unique_Subject`.

`TipcGetGmdDir` uses `mkdir` as needed to create the directory structure if it does not already exist. If any directories are created, their protection is set to be writable by all users. If world-writable directories are a security risk for a site, create the necessary directories ahead of time with the desired permissions. Read and write access to this directory is required for all authorized users. GMD files are created with the same permission as the directory they are stored in; this allows for easier security configuration.

See Also `TipcSrvConnGmdFileCreate`; see the *TIBCO SmartSockets Utilities* for information on `TutGetSocketDir`.

Examples This example uses `TipcGetGmdDir` to check if this process is configured properly for file-based GMD:

```
if (TipcGetGmdDir() != NULL) {
    TutOut("This process is configured properly for file-based
GMD.\n");
}
else {
    if (TutErrNumGet() == T_ERR_VAL_REQUIRED) {
        TutOut("Unique_Subject must be set to a non-default value.\n");
    }
    else if (TutErrNumGet() == T_ERR_C) {
        TutOut("Creation of GMD dirs failed.\n");
        TutPerror("TipcGetGmdDir");
    }
    else {
        return; /* error */
    }
}
```

TipcInitCommands

Name	TipcInitCommands — create SmartSockets IPC commands
Synopsis	T_BOOL TipcInitCommands()
Arguments	None
Return Values	TRUE if the SmartSockets IPC commands were successfully created, FALSE otherwise.
Diagnostics	If TipcInitCommands fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the SmartSockets IPC commands have already been created
Description	TipcInitCommands adds these IPC-specific commands to the SmartSockets command interface: - connect — create connection to RTserver - disconnect — destroy connection to RTserver - subscribe — start subscribing to one or more subjects - unsubscribe — stop subscribing to one or more subjects TipcInitCommands is useful for user-defined processes.
Caution	The commands created with TipcInitCommands (connect, disconnect, subscribe, and unsubscribe) act on the global connection unless received via a CONTROL message, in which case the command is applied to the connection on which it was received.
See Also	None
Examples	This example adds the IPC commands, connects to RTserver, and subscribes to the subject /foo: <pre>if (!TipcInitCommands()) { return; /* error */ } TutCommandParseStr("connect"); TutCommandParseStr("subscribe /foo");</pre>

TipcInitThreads

Name	TipcInitThreads — enable thread-safety on supported platforms
Synopsis	<code>T_BOOL TipcInitThreads()</code>
Arguments	None
Return Values	TRUE if threads are supported for the operating system, FALSE if not.
Diagnostics	If TipcInitThreads fails, it returns FALSE and sets the global SmartSockets error number to: • <code>T_ERR_TYPE_MISMATCH</code> — the platform does not support threads
Description	TipcInitThreads initializes the thread API and turns on internal thread synchronization calls within the utility and IPC libraries. Programs that call SmartSockets API functions from more than one thread must first call this function, even if they do not use any of the other thread API functions. This protects the integrity of the library's internal data structures. Single-threaded programs do not need to call TipcInitThreads, and indeed should avoid doing so for optimum performance. The added overhead of internal thread synchronization calls is not needed for single-threaded programs.
Caution	If threads are to be used within a program, TipcInitThreads must be called before any other SmartSockets function. The <code>T_ENTRY</code> declaration specifier is required in the definition of all thread entry functions as well as their prototypes.
See Also	None
Examples	This example enables the thread-safety features of SmartSockets and checks to ensure that they are supported on the current platform: <pre>int main() { if (!TipcInitThreads()) { return; /* error */ } }</pre>

TipcLbModeToStr

Name	TipcLbModeToStr — convert a message load balancing mode to a string
Synopsis	<pre>T_BOOL TipcLbModeToStr(<i>lb_mode</i>, <i>lb_mode_str_return</i>) T_IPC_LB_MODE <i>lb_mode</i>; T_STR *<i>lb_mode_str_return</i>;</pre>
Arguments	<i>lb_mode</i> — load balancing mode to convert <i>lb_mode_str_return</i> — storage for string
Return Values	TRUE if the load balancing mode was successfully converted to a string, FALSE otherwise.
Diagnostics	If TipcLbModeToStr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>lb_mode</i> was not a valid T_IPC_LB_MODE • T_ERR_NULL_PTR — <i>lb_mode_str_return</i> was null.
Description	TipcLbModeToStr converts a message load balancing mode (such as T_IPC_LB_SORTED) to a lower-case string (such as sorted).
Caution	If TipcLbModeToStr returns FALSE, it does not store a value in <i>lb_mode_str_return</i> . The value stored in <i>lb_mode_str_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcStrToLbMode
Examples	This example prints the load balancing mode of a message: <pre>T_IPC_LB_MODE lb_mode; T_STR lb_mode_str; if (!TipcMsgGetLbMode(msg, &lb_mode)) { return; /* error */ } if (!TipcLbModeToStr(lb_mode, &lb_mode_str)) { return; /* error */ } TutOut("Message has load balancing mode %s (%d).\n", lb_mode_str, lb_mode);</pre>

Chapter 5 **TipcMon***

This chapter covers the TipcMon API functions. For information on the TipcSrvMon API functions that handle multiple connections, see Chapter 15, TipcSrvMon*. This chapter covers these TipcMon API functions:

- *TipcMonClientBufferGet Watch*
- *TipcMonClientBufferPoll*
- *TipcMonClientBufferSet Watch*
- *TipcMonClientCbPoll*
- *TipcMonClientCongestionGet Watch*
- *TipcMonClientCongestionSet Watch*
- *TipcMonClientCpuPoll*
- *TipcMonClientExtPoll*
- *TipcMonClientGeneralPoll*
- *TipcMonClientInfoPoll*
- *TipcMonClientMsgRecvGet Watch*
- *TipcMonClientMsgRecvSet Watch*
- *TipcMonClientMsgSendGet Watch*
- *TipcMonClientMsgSendSet Watch*
- *TipcMonClientMsgTrafficPoll*
- *TipcMonClientMsgTypeExPoll*
- *TipcMonClientMsgTypePoll*
- *TipcMonClientNamesGet Watch*
- *TipcMonClientNamesNumPoll*
- *TipcMonClientNamesPoll*
- *TipcMonClientNamesSet Watch*
- *TipcMonClientOptionPoll*
- *TipcMonClientSubjectExPoll*
- *TipcMonClientSubjectPoll*

- *TipcMonClientSubscribeGet Watch*
- *TipcMonClientSubscribeNumPoll*
- *TipcMonClientSubscribePoll*
- *TipcMonClientSubscribeSet Watch*
- *TipcMonClientTimeGet Watch*
- *TipcMonClientTimePoll*
- *TipcMonClientTimeSet Watch*
- *TipcMonClientVersionPoll*
- *TipcMonExtDelete*
- *TipcMonExtSetBinary*
- *TipcMonExtSetBool*
- *TipcMonExtSetBoolArray*
- *TipcMonExtSetInt2*
- *TipcMonExtSetInt2Array*
- *TipcMonExtSetInt4*
- *TipcMonExtSetInt4Array*
- *TipcMonExtSetInt8*
- *TipcMonExtSetInt8Array*
- *TipcMonExtSetReal4*
- *TipcMonExtSetReal4Array*
- *TipcMonExtSetReal8*
- *TipcMonExtSetReal8Array*
- *TipcMonExtSetReal16*
- *TipcMonExtSetReal16Array*
- *TipcMonExtSetStr*
- *TipcMonExtSetStrArray*
- *TipcMonExtSetUtf8*
- *TipcMonExtSetUtf8Array*
- *TipcMonGetIdentStr*
- *TipcMonPrint Watch*

- *TipcMonProjectNamesGet Watch*
- *TipcMonProjectNamesPoll*
- *TipcMonProjectNamesSet Watch*
- *TipcMonServerBufferPoll*
- *TipcMonServerCongestionGet Watch*
- *TipcMonServerCongestionSet Watch*
- *TipcMonServerConnGet Watch*
- *TipcMonServerConnPoll*
- *TipcMonServerConnSet Watch*
- *TipcMonServerCpuPoll*
- *TipcMonServerGeneralPoll*
- *TipcMonServerMaxClientLicensesGet Watch*
- *TipcMonServerMaxClientLicensesSet Watch*
- *TipcMonServerMsgTrafficExPoll*
- *TipcMonServerMsgTrafficPoll*
- *TipcMonServerNamesGet Watch*
- *TipcMonServerNamesPoll*
- *TipcMonServerNamesSet Watch*
- *TipcMonServerOptionPoll*
- *TipcMonServerRoutePoll*
- *TipcMonServerStartTimePoll*
- *TipcMonServerTimePoll*
- *TipcMonServerVersionPoll*
- *TipcMonSetIdentStr*
- *TipcMonSubjectNamesGet Watch*
- *TipcMonSubjectNamesPoll*
- *TipcMonSubjectNamesSet Watch*
- *TipcMonSubjectSubscribeGet Watch*
- *TipcMonSubjectSubscribePoll*
- *TipcMonSubjectSubscribeSet Watch*

TipcMonClientBufferGetWatch

Name	TipcMonClientBufferGetWatch — determine if this RTclient is watching message-related buffer information in an RTclient
Synopsis	<pre>T_BOOL TipcMonClientBufferGetWatch(<i>client_name</i>, <i>watch_status_return</i>) T_STR <i>client_name</i>; T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope) <i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the RTclient buffer information watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonClientBufferGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>client_name</i> or <i>watch_status_return</i> was null
Description	TipcMonClientBufferGetWatch gets whether this RTclient is watching message-related buffer information in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_BUFFER_STATUS message each time the message queue for the connection to RTserver changes in the watched RTclient. Each MON_CLIENT_BUFFER_STATUS message contains five fields: • a STR field containing the name of the RTclient • an INT4 field containing the number of queued incoming message in the RTclient’s connection to RTserver • an INT4 field containing the number of bytes of data in the queued incoming messages in the RTclient’s connection to RTserver • an INT4 field containing the number of bytes of data in the read buffer in the RTclient’s connection to RTserver • an INT4 field containing the number of bytes of data in the write buffer in the RTclient’s connection to RTserver

To set whether this RTclient is watching RTclient buffer information, use `TipcMonClientBufferSetWatch`. To poll once for RTclient buffer information, use `TipcMonClientBufferPoll`.

For an example of how `MON_CLIENT_BUFFER_STATUS` messages can be used, see the RTmon GDI Watch Client Buffer window.

Caution If `TipcMonClientBufferGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

See Also `TipcMonClientBufferPoll`, `TipcMonClientBufferSetWatch`

Examples This example uses `TipcMonClientBufferGetWatch` to get the status of watching buffer information in the primary RTclient:

```
T_BOOL watch_status;

if (!TipcMonClientBufferGetWatch("primary", &watch_status)) {
    return; return; /*error*/
}
TutOut("This RTclient is %s RTclient buffer information in ",
      watch_status ? "watching" : "not watching");
TutOut("primary.\n");
```

TipcMonClientBufferPoll

Name	TipcMonClientBufferPoll — poll once for RTclient message-related buffer information
Synopsis	<pre>T_BOOL TipcMonClientBufferPoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll for buffer information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)
Return Values	TRUE if the poll for RTclient buffer information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientBufferPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientBufferPoll polls for RTclient message-related buffer information by sending a MON_CLIENT_BUFFER_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_BUFFER_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_CLIENT_BUFFER_POLL_RESULT message contains five fields: • a STR field containing the name of the RTclient • an INT4 field containing the number of queued incoming message in the RTclient’s connection to RTserver • an INT4 field containing the number of bytes of data in the queued incoming messages in the RTclient’s connection to RTserver • an INT4 field containing the number of bytes of data in the read buffer in the RTclient’s connection to RTserver • an INT4 field containing the number of bytes of data in the write buffer in the RTclient’s connection to RTserver

There are multiple responses only if *client_name* is a wildcard or T_IPC_MON_ALL.

To watch RTclient buffer information, use TipcMonClientBufferSetWatch.

Caution TipcMonClientBufferPoll sends a MON_CLIENT_BUFFER_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client_name* does not exist; the poll is silently dropped.

See Also TipcMonClientBufferGetWatch, TipcMonClientBufferSetWatch, TipcSrvMsgWrite

Examples This example uses TipcMonClientBufferPoll to poll for buffer information in all RTclients (including this one), creates a callback to process the incoming MON_CLIENT_BUFFER_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_buffer_poll_result -- process a MON_CLIENT_BUFFER_POLL_RESULT
message */
void T_ENTRY process_mon_client_buffer_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_INT4 msg_queue_count;
    T_INT4 msg_queue_byte_count;
    T_INT4 read_buffer_count;
    T_INT4 write_buffer_count;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg,
        T_IPC_FT_STR, &client_name,
        T_IPC_FT_INT4, &msg_queue_count,
        T_IPC_FT_INT4, &msg_queue_byte_count,
        T_IPC_FT_INT4, &read_buffer_count,
        T_IPC_FT_INT4, &write_buffer_count,
        NULL)) {
        return; /* error */
    }
}
```

```

    TutOut("Got CLIENT_BUFFER poll result.\n");
    TutOut("RTclient name = %s\n", client_name);
    TutOut("msg queue count = %d\n", msg_queue_count);
    TutOut("msg queue byte count = %d\n", msg_queue_byte_count);
    TutOut("read buffer count = %d\n", read_buffer_count);
    TutOut("write buffer count = %d\n", write_buffer_count);
} /* process_mon_client_buffer_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientBufferPoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_BUFFER_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
    process_mon_client_buffer_poll_result,
    NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonClientBufferSetWatch

Name	TipcMonClientBufferSetWatch — start or stop watching RTclient message-related buffer information
Synopsis	<pre>T_BOOL TipcMonClientBufferSetWatch(<i>client_name</i>, <i>watch_status</i>) T_STR <i>client_name</i>; T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to set watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope) <i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the RTclient buffer information watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonClientBufferSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> was null • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonClientBufferSetWatch sets whether this RTclient is watching message-related buffer information in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. TipcMonClientBufferSetWatch saves the watch status and sends a MON_CLIENT_BUFFER_SET_WATCH message to the <i>client_name</i> RTclient. The watch status controls whether the RTclient receives a MON_CLIENT_BUFFER_STATUS message each time the message queue for the connection to RTserver changes in the watched RTclient. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.

Each `MON_CLIENT_BUFFER_STATUS` message contains five fields:

- a `STR` field containing the name of the `RTclient`
- an `INT4` field containing the number of queued incoming message in the `RTclient`'s connection to `RTserver`
- an `INT4` field containing the number of bytes of data in the queued incoming messages in the `RTclient`'s connection to `RTserver`
- an `INT4` field containing the number of bytes of data in the read buffer in the `RTclient`'s connection to `RTserver`
- an `INT4` field containing the number of bytes of data in the write buffer in the `RTclient`'s connection to `RTserver`

In addition to sending a `MON_CLIENT_BUFFER_STATUS` message each time the message queue changes, the watched `RTclient` also sends an initial `MON_CLIENT_BUFFER_STATUS` message when the watch status is set to `TRUE`, so that the `RTclient` does not have to wait for the next message queue change to receive the first `MON_CLIENT_BUFFER_STATUS` message.

To determine if this `RTclient` is watching `RTclient` buffer information, use `TipcMonClientBufferGetWatch`. To poll once for `RTclient` buffer information, use `TipcMonClientBufferPoll`.

For an example of how `MON_CLIENT_BUFFER_STATUS` messages can be used, see the `RTmon GDI Watch Client Buffer` window.

Caution `TipcMonClientBufferSetWatch` sends a `MON_CLIENT_BUFFER_SET_WATCH` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no initial status message if the `RTclient` named *client_name* does not exist.

Buffer changes caused by `MON_CLIENT_BUFFER_STATUS`, `MON_CLIENT_MSG_RECV_STATUS`, and `MON_CLIENT_MSG_SEND_STATUS` message types cannot be watched and are silently ignored by the internal monitoring callbacks that generate outgoing `MON_CLIENT_BUFFER_STATUS` messages. The callbacks ignore buffer changes due to these message types to prevent infinite loops.

See Also `TipcMonClientBufferGetWatch`, `TipcMonClientBufferPoll`

Examples This example uses TipcMonClientBufferSetWatch to start watching buffer information in the primary RTclient and creates a callback to process the incoming MON_CLIENT_BUFFER_STATUS messages:

```
/* ===== */
/*..process_mon_client_buffer_status -- process a MON_CLIENT_BUFFER_STATUS message */

void T_ENTRY process_mon_client_buffer_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_INT4 msg_queue_count;
    T_INT4 msg_queue_byte_count;
    T_INT4 read_buffer_count;
    T_INT4 write_buffer_count;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg,
                     T_IPC_FT_STR, &client_name,
                     T_IPC_FT_INT4, &msg_queue_count,
                     T_IPC_FT_INT4, &msg_queue_byte_count,
                     T_IPC_FT_INT4, &read_buffer_count,
                     T_IPC_FT_INT4, &write_buffer_count,
                     NULL)) {
        return; /* error */
    }

    TutOut("Got CLIENT_BUFFER status.\n");
    TutOut("RTclient name = %s\n", client_name);
    TutOut("msg queue count = %d\n", msg_queue_count);
    TutOut("msg queue byte count = %d\n", msg_queue_byte_count);
    TutOut("read buffer count = %d\n", read_buffer_count);
    TutOut("write buffer count = %d\n", write_buffer_count);
} /* process_mon_client_buffer_status */

/* ===== */
```

```

/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcMonClientBufferSetWatch("primary", TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_BUFFER_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_mon_client_buffer_status,
    NULL)
    == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonClientCbPoll

Name	TipcMonClientCbPoll — poll once for RTclient callback information
Synopsis	<pre>T_BOOL TipcMonClientCbPoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll for callback information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)
Return Values	TRUE if the poll for RTclient callback information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientCbPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientCbPoll polls for RTclient callback information by sending a MON_CLIENT_CB_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_CB_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_CLIENT_CB_POLL_RESULT message contains nine fields: • a STR field containing the name of the RTclient • an INT4 field containing the number of global read callbacks of the polled RTclient connection • an INT4 field containing the number of global write callbacks of the polled RTclient connection • an INT4 field containing the number of global process callbacks of the polled RTclient connection • an INT4 field containing the number of global queue callbacks of the polled RTclient connection • an INT4 field containing the number of default callbacks of the polled RTclient connection

- an INT4 field containing the number of error callbacks of the polled RTclient connection
- an INT4 field containing the number of server create callbacks of the polled RTclient connection
- an INT4 field containing the number of server destroy callbacks of the polled RTclient connection

There are multiple responses only if *client_name* is a wildcard or T_IPC_MON_ALL.

Caution TipcMonClientCbPoll sends a MON_CLIENT_CB_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client_name* does not exist; the poll is silently dropped.

RTclient callback information can only be polled, not watched.

There is no way to poll for RTserver callback information.

See Also TipcMonClientMsgTrafficPoll, TipcSrvMsgWrite, TipcSrvGetAutoFlushSize

Examples This example uses TipcMonClientCbPoll to poll for callback information in all RTclients, including this one, creates a callback to process the incoming MON_CLIENT_CB_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_cb_poll_result -- process a MON_CLIENT_CB_POLL_RESULT message
*/
void T_ENTRY process_mon_client_cb_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_INT4 num_global_read_cb;
    T_INT4 num_global_write_cb;
    T_INT4 num_global_process_cb;
    T_INT4 num_global_queue_cb;
    T_INT4 num_default_cb;
    T_INT4 num_error_cb;
    T_INT4 num_server_create_cb;
    T_INT4 num_server_destroy_cb;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }
}
```



```

/* get the fields from the message */
if (!TipcMsgRead(data->msg,
    T_IPC_FT_STR, &client_name,
    T_IPC_FT_INT4, &num_global_read_cb,
    T_IPC_FT_INT4, &num_global_write_cb,
    T_IPC_FT_INT4, &num_global_process_cb,
    T_IPC_FT_INT4, &num_global_queue_cb,
    T_IPC_FT_INT4, &num_default_cb,
    T_IPC_FT_INT4, &num_error_cb,
    T_IPC_FT_INT4, &num_server_create_cb,
    T_IPC_FT_INT4, &num_server_destroy_cb,
    NULL)) {

return; /* error */
}

TutOut("Got CLIENT_CB poll result.\n");
TutOut("RTclient name = %s\n", client_name);
TutOut("num global read cb = %d\n", num_global_read_cb);
TutOut("num global write cb = %d\n", num_global_write_cb);
TutOut("num global process cb = %d\n", num_global_process_cb);
TutOut("num global queue cb = %d\n", num_global_queue_cb);
TutOut("num default cb = %d\n", num_default_cb);
TutOut("num error cb = %d\n", num_error_cb);
TutOut("num server create cb = %d\n", num_server_create_cb);
TutOut("num server destroy cb = %d\n", num_server_destroy_cb);
} /* process_mon_client_cb_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientCbPoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_CB_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
    process_mon_client_cb_poll_result,
    NULL) == NULL) {

    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonClientCongestionGetWatch

Name	TipcMonClientCongestionGetWatch — determine if the RTclient is watching a client's write buffer for congestion
Synopsis	<pre>T_BOOL TipcMonClientCongestionGetWatch(<i>client_name</i>, <i>watch_status_return</i>) T_STR <i>client_name</i>; T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>client_name</i> — name of the RTclient for which a watch is being checked <i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonClientCongestionGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>client_name</i> or <i>watch_status_return</i> was null
Description	TipcMonClientCongestionGetWatch gets whether the RTclient is watching for congestion in the <i>client_name</i> RTclient write buffer. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The <i>watch_status_return</i> argument is set to TRUE if the watch is currently active, FALSE if not.
Caution	TipcMonClientCongestionGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i>. RTclient write buffer congestion can only be watched, not polled.
See Also	TipcMonClientBufferSetWatch, TipcMonClientCongestionSetWatch, TipcMonServerCongestionSetWatch, TipcMonServerCongestionGetWatch
Examples	This example uses TipcMonClientCongestionGetWatch to determine the status of a buffer congestion watch in the primary RTclient: <pre>T_BOOL watch_status; if (!TipcMonClientCongestionGetWatch("primary", &watch_status)) { return; /* error */ } TutOut("This RTclient is %s RTclient buffer congestion in ", watch_status ? "watching" : "not watching"); TutOut("primary.\n");</pre>

TipcMonClientCongestionSetWatch

Name	TipcMonClientCongestionSetWatch — start or stop watching for congestion in an RTclient's write buffer
Synopsis	<pre>T_BOOL TipcMonClientCongestionSetWatch(<i>client_name</i>, <i>high_water</i>, <i>low_water</i>, <i>watch_status</i>) T_STR <i>client_name</i>; T_INT4 <i>high_water</i>; T_INT4 <i>low_water</i>; T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>client_name</i> — name of the RTclient to set the watch status for <i>high_water</i> — the number of pending bytes that must be exceeded in the buffer for it to be considered congested <i>low_water</i> — the quantity that the number of pending bytes must fall below to end buffer congestion <i>watch_status</i> — new watch status (TRUE to start watching or FALSE to stop watching)
Return Values	TRUE if the watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonClientCongestionSetWatch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>client_name</i> was null • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE, or the value of <i>low_water</i> was less than 0 or greater than the value of <i>high_water</i> • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonClientCongestionSetWatch starts or stops the watch of the write buffer in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. TipcMonClientCongestionSetWatch saves the watch status and sends a MON_CLIENT_CONGESTION_SET_WATCH message to <i>client_name</i>. The watch status controls whether the RTclient receives a MON_CLIENT_CONGESTION_STATUS message when the number of bytes in the watched RTclient's write buffer reaches the <i>high_water</i> or <i>low_water</i> threshold.

When the write buffer passes the value of *high_water*, the RTserver sends a `MON_CLIENT_CONGESTION_STATUS` message. The next `MON_CLIENT_CONGESTION_STATUS` message is sent when the number of bytes in the write buffer decreases to the *low_water* number.

If *watch_status* is `FALSE`, the values in *high_water* and *low_water* are ignored.

The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.

Each `MON_CLIENT_CONGESTION_STATUS` message contains five fields:

- an INT4 field containing the current size of the write buffer
- an INT4 field containing the threshold value of the watch (the *high_water* or *low_water* number)
- a STR field containing the *client_name*
- a BOOL field containing which threshold was triggered (`TRUE` for the *high_water* threshold, `FALSE` if the *low_water* threshold was passed)

Once the number of bytes in the write buffer has passed the *high_water* number, RTserver does not send another `MON_CLIENT_CONGESTION_STATUS` message until number of pending bytes in the buffer decreases to the *low_water* number.

Caution TipcMonClientCongestionSetWatch sends a `MON_CLIENT_CONGESTION_SET_WATCH` message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

RTclient write buffer congestion can only be watched, not polled.

See Also TipcMonClientBufferGetWatch, TipcMonClientCongestionGetWatch
TipcMonServerCongestionSetWatch, TipcMonServerCongestionGetWatch

Examples This example uses TipcMonClientCongestionSetWatch to start watching buffer congestion in the *primary_rtie* RTclient with a *high_water* threshold of 8 million bytes and a *low_water* threshold of 2 million bytes, and creates a callback to process the incoming `MON_CLIENT_CONGESTION_STATUS` messages:

```
/* ===== */
/*..process_mon_client_congestion_status - process a
MON_CLIENT_CONGESTION_STATUS message */

void process_mon_client_congestion_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_INT4 size;
```

```

T_INT4 threshold;
T_STR client_name;
T_BOOL high_water;

/* must set current field first */
if (!TipcMsgSetCurrent(data->msg, 0)) {
    return; /* error */
}

/* get the fields from the message */
if (!TipcMsgRead(data->msg,
                 T_IPC_FT_INT4, &size,
                 T_IPC_FT_INT4, &threshold,
                 T_IPC_FT_STR, &client_name,
                 T_IPC_FT_BOOL, &high_water,
                 NULL)) {
    return; /* error */
}

TutOut("Got CLIENT_CONGESTION status.\n");
TutOut("Crossed %s water threshold.\n", high_water ? "high" :
"low");
TutOut("RTclient name = %s\n", client_name);
TutOut("buffer size = %d\n", size);
TutOut("threshold = %d\n", threshold);
} /* process_mon_client_congestion_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientCongestionSetWatch ("primary_rtie", 8000000,
2000000,
                                     TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_CONGESTION_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
process_mon_client_congestion_status,
                          NULL) == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonClientCpuPoll

Name	TipcMonClientCpuPoll — poll for the percentage of CPU time used by an RTclient
Synopsis	<pre>T_BOOL TipcMonClientCpuPoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll CPU usage for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)
Return Values	TRUE if the poll for CPU usage was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientCpuPoll fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>client_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientCpuPoll polls for the percentage of CPU time used by an RTclient by sending a MON_CLIENT_CPU_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_CPU_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. Each MON_CLIENT_CPU_POLL_RESULT contains two fields: • a STR field containing the name of the RTclient • a REAL4 field containing the percentage of CPU time used (between 0.0 and 100.0) There are multiple responses only if <i>client_name</i> is a wildcard or T_IPC_MON_ALL. If the RTclient cannot query the operating system, the value reported in the CPU percentage used field is a negative number.

Caution There is no response message if the RTclient named *client_name* does not exist; the poll is silently dropped.

TipcMonClientCpuPoll sends a MON_CLIENT_CPU_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

RTclient CPU usage information can only be polled, not watched.

See Also TipcMonServerCpuPoll

Examples This example uses TipcMonClientCpuPoll to:

- poll for information in all RTclients, including the RTclient initiating the poll
- create a callback to process incoming MON_CLIENT_CPU_POLL_RESULT messages
- wait up to 10 seconds for the poll results

```
/* ===== */
/*..process_mon_client_cpu_poll_result - process a
MON_CLIENT_CPU_POLL_RESULT message */

void T_ENTRY process_mon_client_cpu_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_REAL4 cpu_utilization;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg,
                     T_IPC_FT_STR, &client_name,
                     T_IPC_FT_REAL4, &cpu_utilization,
                     NULL)) {
        return; /* error */
    }

    TutOut("Got CLIENT_CPU poll result.\n");
    TutOut("RTclient name = %s\n", client_name);
    TutOut("CPU utilization percentage = %f\n", cpu_utilization);
} /* process_mon_client_cpu_poll_result */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientCpuPoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_CPU_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_mon_client_cpu_poll_result,
                          NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming messages) that arrive within the
next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```


TipcMonClientExtPoll

Name	TipcMonClientExtPoll — poll for extension data of an RTclient
Synopsis	<pre>T_BOOL TipcMonClientExtPoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of the RTclient to retrieve extension data from. The value of <i>client_name</i> can be: • the unique subject name of an RTclient • a subject specified with wildcard characters to match multiple RTclients; for example, <code>/stocks/auto/...</code> or <code>/stocksla*/...</code> • <code>T_IPC_MON_ALL</code> which specifies to poll all RTclients in the project that match the value of the <code>Monitor_Scope</code> option
Return Values	TRUE if the poll for RTclient extension data was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientExtPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_NULL_PTR</code> — <i>client_name</i> was null • any error number from TipcConnMsgWrite
Description	TipcMonClientExtPoll requests extension data from one or more RTclients by sending a message of type <code>MON_CLIENT_EXT_POLL_CALL</code> to those RTclients. Extension data is defined as any information generated by an RTclient which another RTclient wants to monitor. The extension data is not generated by SmartSockets. One or more RTclients respond to a <code>MON_CLIENT_EXT_POLL_CALL</code> message by returning their data in a message of type <code>MON_CLIENT_EXT_POLL_RESULT</code>.
See Also	Use the necessary TipcMonExt* APIs to define extension data in each RTclient that is requested by TipcMonClientExtPoll.

Examples In this example, RTclient1 copies the value of its balance field (100.0) into the field called account_balance in its MON_CLIENT_EXT_POLL_RESULT message. RTclient2 requests the account information from RTclient 1 with TipcMonClientExtPoll and prints the account balance:

RTclient 1:

```
.
.
.

    T_REAL4 balance=100.0;
    TipcMonExtSetReal4("account_balance",balance);
    TipcSrvMainLoop(T_TIMEOUT_FOREVER);
    TipcMonExtDelete("account_balance");.
.
.
.
```

RTclient 2

```
.
.
.

void T_ENTRY ext_poll_result_cb
(
    T_IPC_CONN conn,
    T_IPC_CONN_MSG_CB_DATA data,
    T_PTR arg
)
{
    T_STR    client;
    T_REAL4  account_balance = 0;
    T_INT4   num_named_fields = 0;
    T_BOOL   has_account_balance;

    TipcMsgSetCurrent(data->msg,0);

    TipcMsgNextStr(data->msg,&client);
    TipcMsgNextInt4(data->msg,&num_named_fields);

    /* Get "account_balance" extension data from RTclient1 */

    has_account_balance = TipcMsgGetNamedReal4(data->msg,
                                                "account_balance",
                                                &account_balance);

    if (T_TRUE == has_account_balance) {
        TutOut("Client %s has an account balance of %f dollars.\n",
              client,
              account_balance);
    }
}
.
.
.
```

```
TipcSrvProcessCbCreate(  
    TipcMtLookupByNum(T_MT_MON_CLIENT_EXT_POLL_RESULT),  
    ext_poll_result_cb,T_NULL);  
TipcMonClientExtPoll(T_IPC_MON_ALL);  
TipcSrvMainLoop(T_TIMEOUT_FOREVER);  
.  
.  
.
```

TipcMonClientGeneralPoll

Name	TipcMonClientGeneralPoll — poll once for RTclient general information
Synopsis	<pre>T_BOOL TipcMonClientGeneralPoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll for general information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)
Return Values	TRUE if the poll for RTclient general information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientGeneralPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientGeneralPoll polls for RTclient general information by sending a MON_CLIENT_GENERAL_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_GENERAL_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_CLIENT_GENERAL_POLL_RESULT message contains sixteen fields: • a STR field containing the name of the RTclient • a STR field containing the process monitoring identifier (defaults to a major type of license, such as Development_RTmon) • a STR field containing the node name the RTclient is running on • a STR field containing the user name the RTclient is running under • an INT4 field containing the PID of the RTclient • a STR field containing the project the RTclient is a part of • a STR field containing the name of the RTserver the RTclient is connected to

- a STR field containing the name of the SmartSockets architecture (such as `sun4_solaris`)
- an INT4 field containing the current operating system `sbrk` value (the edge of the RTclient's virtual memory address space)
- an INT4 field containing the amount of change in the current operating system `sbrk` value (the edge of the process's virtual memory address space) since the RTclient started (actually, since the process's first call to any SmartSockets function)
- an INT2 field containing the integer number format of the RTclient (really of type `T_INT_FORMAT`, but stored in an INT2 field to save space)
- an INT2 field containing the real number format of the RTclient (really of type `T_REAL_FORMAT`, but stored in an INT2 field to save space)
- a STR field containing the fully expanded logical connection name of the connection to RTserver
- an INT4_ARRAY field containing the counted licenses that the RTclient has checked out (really an array of type `T_LICENSE`, but stored in an INT4_ARRAY field for portability)
- a STR_ARRAY field containing the extra licenses that the RTclient has checked out (such as `snmp`)
- a STR_ARRAY field containing the names of the subjects that the RTclient is subscribing to

There are multiple responses only if *client_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTserver general information, use `TipcMonServerGeneralPoll`.

For an example of how `MON_CLIENT_GENERAL_POLL_RESULT` messages can be used, see the RTmon GDI Client Information window.

Caution `TipcMonClientGeneralPoll` sends a `MON_CLIENT_GENERAL_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTclient named *client_name* does not exist; the poll is silently dropped.

RTclient general information can only be polled, not watched.

See Also `TipcMonServerGeneralPoll`, `TipcSrvMsgWrite`

Examples This example uses TipcMonClientGeneralPoll to poll for general information in all RTclients (including this one), creates a callback to process the incoming MON_CLIENT_GENERAL_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_general_poll_result -- process a
MON_CLIENT_GENERAL_POLL_RESULT message */
void T_ENTRY process_mon_client_general_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_STR ident;
    T_STR node_name;
    T_STR user_name;
    T_INT4 pid;
    T_STR project;
    T_STR server_name;
    T_STR arch;
    T_INT4 current_sbrk;
    T_INT4 delta_sbrk;
    T_INT2 int_format;
    T_INT2 real_format;
    T_STR logical_conn_name;
    T_INT4 *counted_lic;
    T_INT4 num_counted_lic;
    T_STR *extra_lic;
    T_INT4 num_extra_lic;
    T_STR *subject_subscribe;
    T_INT4 num_subject_subscribe;
    T_INT4 counter; /* loop counter */

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }
}
```

```

/* get the fields from the message */
if (!TipcMsgRead(data->msg,
    T_IPC_FT_STR, &client_name,
    T_IPC_FT_STR, &ident,
    T_IPC_FT_STR, &node_name,
    T_IPC_FT_STR, &user_name,
    T_IPC_FT_INT4, &pid,
    T_IPC_FT_STR, &project,
    T_IPC_FT_STR, &server_name,
    T_IPC_FT_STR, &arch,
    T_IPC_FT_INT4, &current_sbrk,
    T_IPC_FT_INT4, &delta_sbrk,
    T_IPC_FT_INT2, &int_format,
    T_IPC_FT_INT2, &real_format,
    T_IPC_FT_STR, &logical_conn_name,
    T_IPC_FT_INT4_ARRAY, &counted_lic,
    &num_counted_lic,
    T_IPC_FT_STR_ARRAY, &extra_lic,
    &num_extra_lic,
    T_IPC_FT_STR_ARRAY, &subject_subscribe,
    &num_subject_subscribe,
    NULL)) {
    return; /* error */
}

TutOut("Got CLIENT_GENERAL poll result.\n");
TutOut("RTclient name = %s\n", client_name);
TutOut("ident = %s\n", ident);
TutOut("node name = %s\n", node_name);
TutOut("user name = %s\n", user_name);
TutOut("pid = %d\n", pid);
TutOut("project = %s\n", project);
TutOut("RTserver name = %s\n", server_name);
TutOut("architecture = %s\n", arch);
TutOut("current sbrk value = %d\n", current_sbrk);
TutOut("delta sbrk value = %d\n", delta_sbrk);
TutOut("int format = %d\n", int_format);
TutOut("real format = %d\n", real_format);
TutOut("logical conn name = %s\n", logical_conn_name);
TutOut("counted licenses\n");
for (counter = 0; counter < num_counted_lic; counter++) {
    TutOut(" %d\n", counted_lic[counter]);
}
TutOut("extra licenses\n");
for (counter = 0; counter < num_extra_lic; counter++) {
    TutOut(" %s\n", extra_lic[counter]);
}
TutOut("subjects being subscribed to\n");
for (counter = 0; counter < num_subject_subscribe; counter++) {
    TutOut(" %s\n", subject_subscribe[counter]);
}
} /* process_mon_client_general_poll_result */

```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientGeneralPoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_GENERAL_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
                           process_mon_client_general_poll_result,
                           NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```


TipcMonClientInfoPoll

Name	TipcMonClientInfoPoll — poll for RTclient information
Synopsis	<pre>T_BOOL TipcMonClientInfoPoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)
Return Values	TRUE if the poll was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientInfoPoll fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>client_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientInfoPoll polls for RTclient information by sending a MON_CLIENT_INFO_POLL_CALL message to the RTserver. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The RTserver responds by sending back a MON_CLIENT_INFO_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. Each MON_CLIENT_INFO_POLL_RESULT message contains 15 fields: • a STR field containing the name of the RTclient • a STR field containing the process monitoring identifier (defaults to a major type of license, such as Development_RTmon) • a STR field containing the node name the RTclient is running on • a STR field containing the user name the RTclient is running under • an INT4 field containing the PID of the RTclient • a STR field containing the project the RTclient is a part of • a STR field containing the name of the RTserver the RTclient is connected to • a STR field containing the name of the platform (such as sun4_solaris) • an INT4 field containing the current operating system sbrk value (the edge of the RTclient's virtual memory address space)

- an INT4 field containing the amount of change in the current operating system `sbrk` value (the edge of the process's virtual memory address space) since the RTclient started (actually, since the process's first call to any SmartSockets function)
- an INT2 field containing the integer number format of the RTclient (really of type `T_INT_FORMAT`, but stored in an INT2 field to save space)
- an INT2 field containing the real number format of the RTclient (really of type `T_REAL_FORMAT`, but stored in an INT2 field to save space)
- a STR field containing the fully expanded logical connection name of the connection to RTserver
- and INT4 field containing the number of subjects the RTclient is subscribed to
- a REAL4 field containing the percentage of CPU time used (between 0.0 and 100.0) for the RTclient

There are multiple responses only if *client_name* is a wildcard or `T_IPC_MON_ALL`.

If the RTclient cannot query the operating system, the value reported in the CPU percentage used field is a negative number.

Caution There is no response message if the RTclient named *client_name* does not exist; the poll is silently dropped.

`TipcMonClientInfoPoll` sends a `MON_CLIENT_INFO_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

RTclient information can only be polled, not watched.

See Also `TipcMonClientGeneralPoll`

Examples This example uses TipcMonClientInfoPoll to:

- poll for information in all RTclients, including the RTclient initiating the poll
- create a callback to process incoming MON_CLIENT_INFO_POLL_RESULT messages
- wait up to 10 seconds for the poll results

```
/* ===== */
/*..process_mon_client_info_poll_result – process a
MON_CLIENT_INFO_POLL_RESULT message */

void T_ENTRY process_mon_client_info_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_STR ident;
    T_STR node_name;
    T_STR user_name;
    T_INT4 pid;
    T_STR project;
    T_STR server_name;
    T_STR arch;
    T_INT4 current_sbrk;
    T_INT4 delta_sbrk;
    T_INT2 int_format;
    T_INT2 real_format;
    T_STR logical_conn_name;
    T_INT4 num_subject_subscribe;
    T_REAL4 cpu_utilization;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg,
        T_IPC_FT_STR, &client_name,
        T_IPC_FT_STR, &ident,
        T_IPC_FT_STR, &node_name,
        T_IPC_FT_STR, &user_name,
        T_IPC_FT_INT4, &pid,
        T_IPC_FT_STR, &project,
        T_IPC_FT_STR, &server_name,
        T_IPC_FT_STR, &arch,
        T_IPC_FT_INT4, &current_sbrk,
        T_IPC_FT_INT4, &delta_sbrk,
        T_IPC_FT_INT2, &int_format,
        T_IPC_FT_INT2, &real_format,
        T_IPC_FT_STR, &logical_conn_name,
```

```

        T_IPC_FT_INT4, &num_subject_subscribe,
        T_IPC_FT_REAL4, &cpu_utilization,
        NULL)) {
    return; /* error */
}

TutOut("Got CLIENT_INFO poll result.\n");
TutOut("RTclient name = %s\n", client_name);
TutOut("ident = %s\n", ident);
TutOut("node name = %s\n", node_name);
TutOut("user name = %s\n", user_name);
TutOut("pid = %d\n", pid);
TutOut("project = %s\n", project);
TutOut("RTserver name = %s\n", server_name);
TutOut("architecture = %s\n", arch);
TutOut("current sbrk value = %d\n", current_sbrk);
TutOut("delta sbrk value = %d\n", delta_sbrk);
TutOut("int format = %d\n", int_format);
TutOut("real format = %d\n", real_format);
TutOut("logical conn name = %s\n", logical_conn_name);
TutOut("number of subject subscribes = %d\n",
num_subject_subscribe);
TutOut("CPU utilization percentage = %f\n", cpu_utilization);
} /* process_mon_client_info_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientInfoPoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_INFO_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
process_mon_client_info_poll_result,
NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming messages) that arrive within the
next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonClientMsgRecvGetWatch

Name	TipcMonClientMsgRecvGetWatch — determine if this RTclient is watching received messages in an RTclient
Synopsis	<pre>T_BOOL TipcMonClientMsgRecvGetWatch(<i>client_name</i>, <i>msg_type_name</i>, <i>watch_status_return</i>) T_STR <i>client_name</i>; T_STR <i>msg_type_name</i>; T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope) <i>msg_type_name</i> — name of message type to get watch status for (use T_IPC_MON_ALL to indicate a watch of all message types) <i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the RTclient received messages watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonClientMsgRecvGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>client_name</i>, <i>msg_type_name</i>, or <i>watch_status_return</i> was null
Description	TipcMonClientMsgRecvGetWatch gets whether this RTclient is watching received messages in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_MSG_RECV_STATUS message each time a received message is inserted into or deleted from the message queue (for the connection to RTserver) in the watched RTclient. If <i>msg_type_name</i> is T_IPC_MON_ALL, then all message types in the watched RTclient are watched. Each MON_CLIENT_MSG_RECV_STATUS message contains six fields: - a STR field containing the name of the RTclient - a MSG field containing the received message - an INT4 field containing whether the message was just inserted into or deleted from the connection's message queue (really of type T_BOOL); a value of TRUE means the message was inserted, and a value of FALSE means the message was deleted

- an INT4 field containing the position in the queue where the received message was just inserted or deleted (a value of 0 means the front of the queue)
- a BINARY field containing a unique 8-byte identifier for the received message; this field is useful for processes that need to accurately track the entire contents of the connection's message queue, such as the RTmon GDI
- a BINARY field containing an array of unique 8-byte identifiers for all messages in the connection's message queue; this field is useful for processes that need to accurately track the entire contents of the connection's message queue, such as the RTmon GDI

To set whether this RTclient is watching RTclient received messages, use `TipcMonClientMsgRecvSetWatch`.

For an example of how `MON_CLIENT_MSG_RECV_STATUS` messages can be used, see the RTmon GDI Watch Messages Received window.

Caution If `TipcMonClientMsgRecvGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

RTclient received messages can only be watched, not polled.

See Also `TipcMonClientMsgRecvSetWatch`

Examples This example uses `TipcMonClientMsgRecvGetWatch` to get the status of watching all received messages in the primary RTclient:

```
T_BOOL watch_status;

if (!TipcMonClientMsgRecvGetWatch("primary", T_IPC_MON_ALL,
                                   &watch_status)) {
    return; /* error */
}
TutOut("This process is %s all messages received in primary.\n",
       watch_status ? "watching" : "not watching");
```

TipcMonClientMsgRecvSetWatch

Name	TipcMonClientMsgRecvSetWatch — start or stop watching RTclient received messages
Synopsis	<pre>T_BOOL TipcMonClientMsgRecvSetWatch(<i>client_name</i>, <i>msg_type_name</i>, <i>watch_status</i>) T_STR <i>client_name</i>; T_STR <i>msg_type_name</i>; T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to set watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope) <i>msg_type_name</i> — name of message type to set watch status for (use T_IPC_MON_ALL to indicate a watch of all message types) <i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the RTclient received watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonClientMsgRecvSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> or <i>msg_type_name</i> was null • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonClientMsgRecvSetWatch sets whether this RTclient is watching received messages in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. TipcMonClientMsgRecvSetWatch saves the watch status and sends a MON_CLIENT_MSG_RECV_SET_WATCH message to the <i>client_name</i> RTclient. The watch status controls whether the RTclient receives a MON_CLIENT_MSG_RECV_STATUS message each time a received message is inserted or deleted from the message queue for the connection to RTserver in the watched RTclient. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. If <i>msg_type_name</i> is T_IPC_MON_ALL, then all message types in the watched RTclient are watched.

Each `MON_CLIENT_MSG_RECV_STATUS` message contains six fields:

- a `STR` field containing the name of the `RTclient`
- a `MSG` field containing the received message
- an `INT4` field containing whether the message was just inserted into or deleted from the connection's message queue (really of type `T_BOOL`); a value of `TRUE` means the message was inserted, a value of `FALSE` means the message was deleted
- an `INT4` field containing the position in the queue where the received message was just inserted or deleted (a value of 0 means the front of the queue)
- a `BINARY` field containing a unique 8-byte identifier for the received message; this field is useful for processes that need to accurately track the entire contents of the connection's message queue, such as the `RTmon GDI`
- a `BINARY` field containing an array of unique 8-byte identifiers for all messages in the connection's message queue; this field is useful for processes that need to accurately track the entire contents of the connection's message queue, such as the `RTmon GDI`

In addition to sending a `MON_CLIENT_MSG_RECV_STATUS` message each time there is a message queue change, the watched `RTclient` also sends an initial `MON_CLIENT_MSG_RECV_STATUS` message when the watch status is set to `TRUE`, so that the `RTclient` does not have to wait for the next message queue change to receive the first `MON_CLIENT_MSG_RECV_STATUS` message.

To see if this `RTclient` is watching `RTclient` received messages, use `TipcMonClientMsgRecvGetWatch`.

For an example of how `MON_CLIENT_MSG_RECV_STATUS` messages can be used, see the `RTmon GDI Watch Messages Received` window.

Caution `TipcMonClientMsgRecvSetWatch` sends a `MON_CLIENT_MSG_RECV_SET_WATCH` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no initial status message if the `RTclient` named *client_name* or the message type named *msg_type_name* does not exist.

`RTclient` received messages can only be watched, not polled.

The `MON_CLIENT_BUFFER_STATUS`, `MON_CLIENT_MSG_RECV_STATUS`, and `MON_CLIENT_MSG_SEND_STATUS` message types cannot be watched and are silently ignored by the internal monitoring callbacks that generate outgoing `MON_CLIENT_MSG_RECV_STATUS` messages. The callbacks ignore these message types to prevent infinite loops.

See Also TipcMonClientMsgRecvGetWatch

Examples This uses TipcMonClientMsgRecvSetWatch to start watching all received messages in the primary RTclient and creates a callback to process the incoming MON_CLIENT_MSG_RECV_STATUS messages:

```
/* ===== */
/*..process_mon_client_msg_recv_status -- process a MON_CLIENT_MSG_RECV_STATUS
message */
void T_ENTRY process_mon_client_msg_recv_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_IPC_MSG recv_msg;
    T_INT4 insert_flag;
    T_INT4 queue_pos;
    T_PTR msg_id;
    T_INT4 msg_id_size;
    T_PTR queue_msg_id_array;
    T_INT4 queue_msg_id_array_size;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg,
        T_IPC_FT_STR, &client_name,
        T_IPC_FT_MSG, &recv_msg,
        T_IPC_FT_INT4, &insert_flag,
        T_IPC_FT_INT4, &queue_pos,
        T_IPC_FT_BINARY, &msg_id, &msg_id_size,
        T_IPC_FT_BINARY, &queue_msg_id_array,
        &queue_msg_id_array_size,
        NULL)) {
        return; /* error */
    }

    TutOut("Got CLIENT_MSG_RECV status.\n");
    TutOut("RTclient name = %s\n", client_name);
    TutOut("recv msg\n");
    if (!TipcMsgPrint(recv_msg, TutOut)) {
        return; /* error */
    }
    TutOut("insert flag = %s\n", insert_flag ? "TRUE" : "FALSE");
    TutOut("queue pos = %d\n", queue_pos);
    /* At this point, msg_id and queue_msg_id_array could be used */
    /* to accurately track any changes to the message queue (as */
    /* RTmon GDI does). */
} /* process_mon_client_msg_recv_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcMonClientMsgRecvSetWatch("primary", T_IPC_MON_ALL,
                                   TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_MSG_RECV_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_mon_client_msg_recv_status,
                           NULL) == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonClientMsgSendGetWatch

Name	TipcMonClientMsgSendGetWatch — determine if this RTclient is watching sent messages in an RTclient
Synopsis	<pre>T_BOOL TipcMonClientMsgSendGetWatch(<i>client_name</i>, <i>msg_type_name</i>, <i>watch_status_return</i>) T_STR <i>client_name</i>; T_STR <i>msg_type_name</i>; T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope) <i>msg_type_name</i> — name of message type to get watch status for (use T_IPC_MON_ALL to indicate a watch of all message types) <i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the RTclient sent messages watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonClientMsgSendGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>client_name</i>, <i>msg_type_name</i>, or <i>watch_status_return</i> was null
Description	TipcMonClientMsgSendGetWatch gets whether this RTclient is watching sent messages in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_MSG_SEND_STATUS message each time a message is sent to RTserver in the watched RTclient. If <i>msg_type_name</i> is T_IPC_MON_ALL, then all message types in the watched RTclient are watched. Each MON_CLIENT_MSG_SEND_STATUS message contains two fields: - a STR field containing the name of the RTclient - a MSG field containing the sent message To set whether this RTclient is watching RTclient sent messages, use TipcMonClientMsgSendSetWatch.

For an example of how `MON_CLIENT_MSG_SEND_STATUS` messages can be used, see the RTmon GDI Watch Messages Sent window.

Caution If `TipcMonClientMsgSendGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

RTclient sent messages can only be watched, not polled.

See Also `TipcMonClientMsgSendSetWatch`

Examples This example uses `TipcMonClientMsgSendGetWatch` to get the status of watching all sent messages in the `primary` RTclient:

```
T_BOOL watch_status;

if (!TipcMonClientMsgSendGetWatch("primary", T_IPC_MON_ALL,
                                   &watch_status)) {
    return; /* error */
}
TutOut("This process is %s all messages sent in primary.\n",
       watch_status ? "watching" : "not watching");
```

TipcMonClientMsgSendSetWatch

Name	TipcMonClientMsgSendSetWatch — start or stop watching RTclient sent messages
Synopsis	<pre>T_BOOL TipcMonClientMsgSendSetWatch(<i>client_name</i>, <i>msg_type_name</i>, <i>watch_status</i>) T_STR <i>client_name</i>; T_STR <i>msg_type_name</i>; T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to set watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope) <i>msg_type_name</i> — name of message type to set watch status for (use T_IPC_MON_ALL to indicate a watch of all message types) <i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the RTclient sent messages watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonClientMsgSendSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> or <i>msg_type_name</i> was null • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonClientMsgSendSetWatch sets whether this RTclient is watching sent messages in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. TipcMonClientMsgSendSetWatch saves the watch status and sends a MON_CLIENT_MSG_SEND_SET_WATCH message to the <i>client_name</i> RTclient. The watch status controls whether the RTclient receives a MON_CLIENT_MSG_SEND_STATUS message each time a message is sent to RTserver in the watched RTclient. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. If <i>msg_type_name</i> is T_IPC_MON_ALL then all message types in the watched RTclient are watched.

Each `MON_CLIENT_MSG_SEND_STATUS` message contains two fields:

- a `STR` field containing the name of the `RTclient`
- a `MSG` field containing the sent message

Unlike most of the other watch categories, the watched `RTclient` does not also send an initial `MON_CLIENT_MSG_SEND_STATUS` message when the watch status is set to `TRUE`. This is because the status message is only sent when a message is sent, and thus there is no current status to use for initialization.

To determine if this `RTclient` is watching `RTclient` sent messages, use `TipcMonClientMsgSendGetWatch`.

For an example of how `MON_CLIENT_MSG_SEND_STATUS` messages can be used, see the `RTmon GDI Watch Messages Sent` window.

Caution `TipcMonClientMsgSendSetWatch` sends a `MON_CLIENT_MSG_SEND_SET_WATCH` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

`RTclient` sent messages can only be watched, not polled.

The `MON_CLIENT_BUFFER_STATUS`, `MON_CLIENT_MSG_RECV_STATUS`, and `MON_CLIENT_MSG_SEND_STATUS` message types cannot be watched and are silently ignored by the internal monitoring callbacks that generate outgoing `MON_CLIENT_MSG_SEND_STATUS` messages. The callbacks ignore these message types to prevent infinite loops.

See Also `TipcMonClientMsgSendGetWatch`

Examples This example uses `TipcMonClientMsgSendSetWatch` to start watching all sent messages in the primary `RTclient` and creates a callback to process the incoming `MON_CLIENT_MSG_SEND_STATUS` messages:

```
/* ===== */
/*..process_mon_client_msg_send_status -- process a MON_CLIENT_MSG_SEND_STATUS
message */

void T_ENTRY process_mon_client_msg_send_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_IPC_MSG send_msg;
```

```

/* must set current field first */
if (!TipcMsgSetCurrent(data->msg, 0)) {
    return; /* error */
}

/* get the fields from the message */
if (!TipcMsgRead(data->msg,
                 T_IPC_FT_STR, &client_name,
                 T_IPC_FT_MSG, &send_msg,
                 NULL)) {
    return; /* error */
}

TutOut("Got CLIENT_MSG_SEND status.\n");
TutOut("RTclient name = %s\n", client_name);
TutOut("send msg\n");
if (!TipcMsgPrint(send_msg, TutOut)) {
    return; /* error */
}
} /* process_mon_client_msg_send_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcMonClientMsgSendSetWatch("primary_rtie", T_IPC_MON_ALL,
                                  TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_MSG_SEND_STATUS);
if (mt == NULL) {
    return; /* error */
}

if(TipcSrvProcessCbCreate(mt,process_mon_client_msg_send_status,
                          NULL) == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonClientMsgTrafficPoll

Name	TipcMonClientMsgTrafficPoll — poll once for RTclient message traffic information
Synopsis	<pre>T_BOOL TipcMonClientMsgTrafficPoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll for message traffic information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)
Return Values	TRUE if the poll for RTclient message traffic information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientMsgTrafficPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientMsgTrafficPoll polls for RTclient message traffic information by sending a MON_CLIENT_MSG_TRAFFIC_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_MSG_TRAFFIC_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_CLIENT_MSG_TRAFFIC_POLL_RESULT message contains these fields: • a STR field containing the name of the RTclient • an INT4 field containing the number of messages the polled RTclient connection has sent • an INT4 field containing the number of messages the polled RTclient connection has received • an INT4 field containing the number of bytes the polled RTclient connection has sent

- an INT4 field containing the number of bytes the polled RTclient connection has received



In version 6.7 and later, the poll also returns these 8-byte variants of the previous fields. When these fields are present, they can be more accurate than their 4-byte counterparts (which may truncate the data):

- an INT8 field containing the number of messages the polled RTclient connection has sent
- an INT8 field containing the number of messages the polled RTclient connection has received
- an INT8 field containing the number of bytes the polled RTclient connection has sent
- an INT8 field containing the number of bytes the polled RTclient connection has received

There are multiple responses only if *client_name* is a wildcard or T_IPC_MON_ALL.

To poll once for message traffic information by message type, use TipcMonClientMsgTypePoll. To poll once for message traffic information by subject, use TipcMonClientSubjectPoll.

To poll once for RTserver message traffic information, use TipcMonServerMsgTrafficPoll.

For an example of how MON_CLIENT_MSG_TRAFFIC_POLL_RESULT messages can be used, see the RTmon GDI Client Message Traffic window.

Caution TipcMonClientMsgTrafficPoll sends a MON_CLIENT_MSG_TRAFFIC_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client_name* does not exist; the poll is silently dropped.

RTclient message traffic information can only be polled, not watched.

See Also TipcMonClientSubjectExPoll, TipcMonClientMsgTypeExPoll, TipcMonServerMsgTrafficExPoll, TipcSrvMsgWrite

Examples This example uses TipcMonClientMsgTrafficPoll to poll for message traffic information in all RTclients (including this one), creates a callback to process the incoming MON_CLIENT_MSG_TRAFFIC_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_msg_traffic_poll_result -- process a
MON_CLIENT_MSG_TRAFFIC_POLL_RESULT message */

void T_ENTRY process_mon_client_msg_traffic_poll_result
(
    T_IPC_CONN conn,
    T_IPC_CONN_PROCESS_CB_DATA data,
    T_CB_ARG arg
)
{
    T_STR client_name;
    T_INT4 total_msg_send_4;
    T_INT4 total_msg_rcv_4;
    T_INT4 total_byte_send_4;
    T_INT4 total_byte_rcv_4;
    T_INT8 total_msg_send_8;
    T_INT8 total_msg_rcv_8;
    T_INT8 total_byte_send_8;
    T_INT8 total_byte_rcv_8;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg, T_IPC_FT_STR, &client_name,
        T_IPC_FT_INT4, &total_msg_send_4,
        T_IPC_FT_INT4, &total_msg_rcv_4,
        T_IPC_FT_INT4, &total_byte_send_4,
        T_IPC_FT_INT4, &total_byte_rcv_4,
        NULL)) {
        return; /* error */
    }

    if (TipcMsgRead(data->msg, T_IPC_FT_INT8, &total_msg_rcv_8,
        T_IPC_FT_INT8, &total_msg_send_8,
        T_IPC_FT_INT8, &total_byte_rcv_8,
        T_IPC_FT_INT8, &total_byte_send_8,
        NULL)) {
        TutOut("Got CLIENT_MSG_TRAFFIC poll result (64-bit data).\n");
        TutOut("RTclient name = %s\n", client_name);
    }
}
```

```

/*
 * IMPORTANT: The format string for T_INT8 differs by platform.
 * The T_INT8_SPEC utility macro supplies the correct format
 * string for each platform.
 */
    TutOut("Total msg recv  = " T_INT8_SPEC "\n",
total_msg_recv_8);
    TutOut("Total msg sent  = " T_INT8_SPEC "\n",
total_msg_send_8);
    TutOut("Total byte recv = " T_INT8_SPEC "\n",
total_byte_recv_8);
    TutOut("Total byte sent = " T_INT8_SPEC "\n",
total_byte_send_8);
}
else {
    TutOut("Got CLIENT_MSG_TRAFFIC poll result (32-bit data).\n");
    TutOut("RTclient name = %s\n", client_name);
    TutOut("Total msg recv  = " T_INT4_SPEC "\n",
total_msg_recv_4);
    TutOut("Total msg sent  = " T_INT4_SPEC "\n",
total_msg_send_4);
    TutOut("Total byte recv = " T_INT4_SPEC "\n",
total_byte_recv_4);
    TutOut("Total byte sent = " T_INT4_SPEC "\n",
total_byte_send_4);
}
} /* process_mon_client_msg_traffic_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_MSG_TRAFFIC_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}
if (TipcSrvProcessCbCreate(mt,
process_mon_client_msg_traffic_poll_result,
NULL) == NULL) {
    return; /* error */
}

/* send the poll request out to RTserver */
if (!TipcMonClientMsgTrafficPoll("/...")) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonClientMsgTypeExPoll

Name TipcMonClientMsgTypeExPoll — poll once for RTclient message type information



This new function supersedes `TipcMonClientMsgTypePoll` for clients that use release 6.7 or later.

Synopsis

```
T_BOOL TipcMonClientMsgTypeExPoll(client_name, msg_type_name)
T_STR client_name;
T_STR msg_type_name;
```

Arguments *client_name* — name of RTclient to poll for message type information (wildcard names allowed, or use `T_IPC_MON_ALL` to indicate a poll of all RTclients matching the value of `Monitor_Scope`)

msg_type_name — name of message type to poll for information (use `T_IPC_MON_ALL` to indicate a poll of all message types)

Return Values TRUE if the poll for RTclient message type information was successfully initiated, FALSE otherwise.

Diagnostics If `TipcMonClientMsgTypeExPoll` fails, it returns FALSE and sets the global SmartSockets error number to one of:

- `T_ERR_NULL_PTR` — *client_name* or *msg_type_name* was null
- any error number from `TipcSrvMsgWrite`

Description `TipcMonClientMsgTypeExPoll` polls for RTclient message type information by sending a `MON_CLIENT_MSG_TYPE_EX_POLL_CALL` message to the *client_name* RTclient. The *client_name* argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or `T_IPC_MON_ALL` to poll all RTclients in the project that match the value of the `Monitor_Scope` option. The polled RTclient responds by sending back a `MON_CLIENT_MSG_TYPE_EX_POLL_RESULT` message. The response may or may not come back quickly, depending on what the polled RTclient is doing. If *msg_type_name* is `T_IPC_MON_ALL`, then all message types in the polled RTclient are polled. The polling RTclient can use functions such as `TipcSrvMsgSearchType` or `TipcSrvMainLoop` to get the response.

Each `MON_CLIENT_MSG_TYPE_EX_POLL_RESULT` message contains one fixed field and fifteen array fields. The fixed field is a STR field containing the name of the RTclient. The array fields are:

- a `STR_ARRAY` field containing the names of the message types
- an `INT4_ARRAY` field containing the message type numbers

- a STR_ARRAY field containing the message type grammars
- an INT4_ARRAY field containing whether the message type priorities are known (each array item is really of type T_BOOL)
- an INT2_ARRAY field containing the message type priorities (not applicable if the corresponding item in the previous array is FALSE)
- an INT4_ARRAY field containing the message type delivery modes (each array item is really of type T_IPC_DELIVERY_MODE)
- an INT4_ARRAY field containing the message type user-defined properties
- an INT4_ARRAY field containing the number of read callbacks the polled RTclient connection has for each message type
- an INT4_ARRAY field containing the number of write callbacks the polled RTclient connection has for each message type
- an INT4_ARRAY field containing the number of process callbacks the polled RTclient connection has for each message type
- an INT4_ARRAY field containing the number of queue callbacks the polled RTclient connection has for each message type
- an INT8_ARRAY field containing the number of messages the polled RTclient connection has sent for each message type
- an INT8_ARRAY field containing the number of messages the polled RTclient connection has received for each message type
- an INT8_ARRAY field containing the number of bytes the polled RTclient connection has sent for each message type
- an INT8_ARRAY field containing the number of bytes the polled RTclient connection has received for each message type

There are multiple responses only if *client_name* is a wildcard or T_IPC_MON_ALL.

To poll once for global message traffic information, use TipcMonClientMsgTrafficExPoll. To poll once for message traffic information by subject, use TipcMonClientSubjectExPoll. To poll once for global callback information, use TipcMonClientCbPoll.

Caution TipcMonClientMsgTypeExPoll sends a MON_CLIENT_MSG_TYPE_EX_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client_name* or the message type named *msg_type_name* does not exist; the poll is silently dropped.

RTclient message type information can only be polled, not watched.

There is no way to poll for RTserver message type information.

See Also TipcMonClientCbPoll, TipcMonClientSubjectExPoll,
TipcMonClientMsgTrafficPoll, TipcSrvMsgWrite

Examples This example uses TipcMonClientMsgTypeExPoll to poll for all message type information in all RTclients (including this one), creates a callback to process the incoming MON_CLIENT_MSG_TYPE_EX_POLL_RESULT messages, and then waits up to 10 seconds for the poll results.

```
/* ===== */
/*..process_mon_client_msg_type_ex_poll_result -- process a
MON_CLIENT_MSG_TYPE_EX_POLL_RESULT message */

void process_mon_client_msg_type_ex_poll_result
(
    T_IPC_CONN conn,
    T_IPC_CONN_PROCESS_CB_DATA data,
    T_CB_ARG arg
)
{
    T_MSG    msg;
    T_STR    client_name;
    T_INT4   counter;
    T_STR    delivery_mode_str = "";

    /* So many arrays... */
    T_STR    *msg_type_name;
    T_INT4   *msg_num;
    T_STR    *grammar;
    T_INT4   *priority_known; /* really (T_BOOL) */
    T_INT2   *priority; /* not applicable if priority_known is FALSE */
    T_INT4   *delivery_mode; /* really T_IPC_DELIVERY_MODE */
    T_INT4   *user_prop;
    T_INT4   *num_read_cb;
    T_INT4   *num_write_cb;
    T_INT4   *num_process_cb;
    T_INT4   *num_queue_cb;
    T_INT8   *total_msg_send;
    T_INT8   *total_msg_recv;
    T_INT8   *total_byte_send;
    T_INT8   *total_byte_recv;

    T_INT4   num_msg_type_name;
    T_INT4   num_msg_num;
    T_INT4   num_grammar;
    T_INT4   num_priority_known;
    T_INT4   num_priority;
    T_INT4   num_delivery_mode;
    T_INT4   num_user_prop;
    T_INT4   num_num_read_cb;
    T_INT4   num_num_write_cb;
```

```

T_INT4  num_num_process_cb;
T_INT4  num_num_queue_cb;
T_INT4  num_msg_send;
T_INT4  num_msg_rcv;
T_INT4  num_byte_send;
T_INT4  num_byte_rcv;

/* must set current field first */
if (!TIPCMsgSetCurrent(data->msg, 0)) {
return; /* error */
}

/* get the fields from the message */
msg = data->msg;
if (!TIPCMsgNextStr(msg, &client_name)
|| !TIPCMsgNextStrArray(msg, &msg_type_name, &num_msg_type_name)
|| !TIPCMsgNextInt4Array(msg, &msg_num, &num_msg_num)
|| !TIPCMsgNextStrArray(msg, &grammar, &num_grammar)
|| !TIPCMsgNextInt4Array(msg, &priority_known,
&num_priority_known)
|| !TIPCMsgNextInt2Array(msg, &priority, &num_priority)
|| !TIPCMsgNextInt4Array(msg, &delivery_mode,
&num_delivery_mode)
|| !TIPCMsgNextInt4Array(msg, &user_prop, &num_user_prop)
|| !TIPCMsgNextInt4Array(msg, &num_read_cb, &num_num_read_cb)
|| !TIPCMsgNextInt4Array(msg, &num_write_cb, &num_num_write_cb)
|| !TIPCMsgNextInt4Array(msg, &num_process_cb,
&num_num_process_cb)
|| !TIPCMsgNextInt4Array(msg, &num_queue_cb, &num_num_queue_cb)
|| !TIPCMsgNextInt8Array(msg, &total_msg_send, &num_msg_send)
|| !TIPCMsgNextInt8Array(msg, &total_msg_rcv, &num_msg_rcv)
|| !TIPCMsgNextInt8Array(msg, &total_byte_send, &num_byte_send)
|| !TIPCMsgNextInt8Array(msg, &total_byte_rcv, &num_byte_rcv))
{
return; /* error */
}
TutOut("Got CLIENT_MSG_TYPE_EX poll result.\n");
TutOut("RTclient name = %s\n", client_name);

/* loop through all groups */
for (counter=0; counter<num_msg_type_name; counter++) {
TutOut("message type name = %s\n", msg_type_name[counter]);
TutOut("num = %d\n", msg_num[counter]);
TutOut("grammar = %s\n", grammar[counter]);
if (priority_known[counter]) {
TutOut("priority = %d\n", priority[counter]);
}
else {
TutOut("priority = unknown\n");
}
TIPCDeliveryModeToStr(delivery_mode[counter],
&delivery_mode_str);
TutOut("delivery mode = %s\n", delivery_mode_str);
TutOut("user prop = " T_INT4_SPEC "\n",
user_prop[counter]);
TutOut("num read cb = " T_INT4_SPEC "\n",

```

```

        num_read_cb[counter]);
    TutOut("num write cb = " T_INT4_SPEC "\n",
        num_write_cb[counter]);
    TutOut("num process cb = " T_INT4_SPEC "\n",
        num_process_cb[counter]);
    TutOut("num queue cb = " T_INT4_SPEC "\n",
        num_queue_cb[counter]);
/*
* IMPORTANT: The format string for T_INT8 differs by platform.
* The T_INT8_SPEC utility macro supplies the correct format
* string for each platform.
*/
    TutOut("total msg send = " T_INT8_SPEC "\n",
        total_msg_send[counter]);
    TutOut("total msg recv = " T_INT8_SPEC "\n",
        total_msg_recv[counter]);
    TutOut("total byte send = " T_INT8_SPEC "\n",
        total_byte_send[counter]);
    TutOut("total byte recv = " T_INT8_SPEC "\n",
        total_byte_recv[counter]);
}

} /* process_mon_client_msg_type_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_MSG_TYPE_EX_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}
if (TipcSrvProcessCbCreate(mt,

process_mon_client_msg_type_ex_poll_result,
                        NULL) == NULL) {
    return; /* error */
}

/* send the poll request out to RTserver */
if (!TipcMonClientMsgTypeExPoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```


TipcMonClientMsgTypePoll

Name TipcMonClientMsgTypePoll — poll once for RTclient message type information



The previous function, `TipcMonClientMsgTypeExPoll`, supersedes this function for all new development. Use this older function only when operating with clients prior to release 6.7. This poll can truncate the statistics data that it returns.

Synopsis `T_BOOL TipcMonClientMsgTypePoll(client_name, msg_type_name)`
 `T_STR client_name;`
 `T_STR msg_type_name;`

Arguments *client_name* — name of RTclient to poll for message type information (wildcard names allowed, or use `T_IPC_MON_ALL` to indicate a poll of all RTclients matching the value of `Monitor_Scope`)

msg_type_name — name of message type to poll for information (use `T_IPC_MON_ALL` to indicate a poll of all message types)

Return Values TRUE if the poll for RTclient message type information was successfully initiated, FALSE otherwise.

Diagnostics If `TipcMonClientMsgTypePoll` fails, it returns FALSE and sets the global SmartSockets error number to one of:

- `T_ERR_NULL_PTR` — *client_name* or *msg_type_name* was null
- any error number from `TipcSrvMsgWrite`

Description `TipcMonClientMsgTypePoll` polls for RTclient message type information by sending a `MON_CLIENT_MSG_TYPE_POLL_CALL` message to the *client_name* RTclient. The *client_name* argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or `T_IPC_MON_ALL` to poll all RTclients in the project that match the value of the `Monitor_Scope` option. The polled RTclient responds by sending back a `MON_CLIENT_MSG_TYPE_POLL_RESULT` message. The response may or may not come back quickly, depending on what the polled RTclient is doing. If *msg_type_name* is `T_IPC_MON_ALL`, then all message types in the polled RTclient are polled. The polling RTclient can use functions such as `TipcSrvMsgSearchType` or `TipcSrvMainLoop` to get the response.

Each `MON_CLIENT_MSG_TYPE_POLL_RESULT` message contains one fixed field and one or more groups of fifteen fields. The fixed field is a STR field containing the name of the RTclient. The group of fifteen fields is:

- a STR field containing the name of the message type
- an INT4 field containing the message type number
- a STR field containing the message type grammar
- an INT4 field containing whether the message type priority is known (really of type `T_BOOL`)
- an INT2 field containing the message type priority (not applicable if the previous field is `FALSE`)
- an INT4 field containing the message type delivery mode (really of type `T_IPC_DELIVERY_MODE`)
- an INT4 field containing the message type user-defined property
- an INT4 field containing the number of read callbacks the polled RTclient connection has for this message type
- an INT4 field containing the number of write callbacks the polled RTclient connection has for this message type
- an INT4 field containing the number of process callbacks the polled RTclient connection has for this message type
- an INT4 field containing the number of queue callbacks the polled RTclient connection has for this message type
- an INT4 field containing the number of messages the polled RTclient connection has sent for this message type
- an INT4 field containing the number of messages the polled RTclient connection has received for this message type
- an INT4 field containing the number of bytes the polled RTclient connection has sent for this message type
- an INT4 field containing the number of bytes the polled RTclient connection has received for this message type

There are multiple responses only if *client_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for global message traffic information, use `TipcMonClientMsgTrafficPoll`. To poll once for message traffic information by subject, use `TipcMonClientSubjectPoll`. To poll once for global callback information, use `TipcMonClientCbPoll`.

Caution TipcMonClientMsgTypePoll sends a `MON_CLIENT_MSG_TYPE_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTclient named *client_name* or the message type named *msg_type_name* does not exist; the poll is silently dropped.

RTclient message type information can only be polled, not watched.

There is no way to poll for RTserver message type information.

See Also `TipcMonClientCbPoll`, `TipcMonClientSubjectPoll`, `TipcMonClientMsgTrafficPoll`, `TipcSrvMsgWrite`

Examples This example uses `TipcMonClientMsgTypePoll` to poll for all message type information in all RTclients (including this one), creates a callback to process the incoming `MON_CLIENT_MSG_TYPE_POLL_RESULT` messages, and then waits up to 10 seconds for the poll results.

```
/* ===== */
/*..process_mon_client_msg_type_poll_result -- process a
MON_CLIENT_MSG_TYPE_POLL_RESULT message */
void T_ENTRY process_mon_client_msg_type_poll_result(
    conn, data, arg)

T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_STR msg_type_name;
    T_INT4 num;
    T_STR grammar;
    T_INT4 priority_known; /* really (T_BOOL) */
    T_INT2 priority; /* not applicable if priority_known is FALSE */
    T_INT4 delivery_mode; /* really T_IPC_DELIVERY_MODE */
    T_INT4 user_prop;
    T_INT4 num_read_cb;
    T_INT4 num_write_cb;
    T_INT4 num_process_cb;
    T_INT4 num_queue_cb;
    T_INT4 total_msg_send;
    T_INT4 total_msg_recv;
    T_INT4 total_byte_send;
    T_INT4 total_byte_recv;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }
}
```

```

/* get first fixed field */
if (!TipcMsgNextStr(data->msg, &client_name)) {
return; /* error */
}
TutOut("Got CLIENT_MSG_TYPE poll result.\n");
TutOut("RTclient name = %s\n", client_name);

/* loop through all groups */
while (TipcMsgRead(data->msg,
    T_IPC_FT_STR, &msg_type_name,
    T_IPC_FT_INT4, &num,
    T_IPC_FT_STR, &grammar,
    T_IPC_FT_INT4, &priority_known,
    T_IPC_FT_INT2, &priority,
    T_IPC_FT_INT4, &delivery_mode,
    T_IPC_FT_INT4, &user_prop,
    T_IPC_FT_INT4, &num_read_cb,
    T_IPC_FT_INT4, &num_write_cb,
    T_IPC_FT_INT4, &num_process_cb,
    T_IPC_FT_INT4, &num_queue_cb,
    T_IPC_FT_INT4, &total_msg_send,
    T_IPC_FT_INT4, &total_msg_rcv,
    T_IPC_FT_INT4, &total_byte_send,
    T_IPC_FT_INT4, &total_byte_rcv,
    NULL)) {
    TutOut("message type name = %s\n", msg_type_name);
    TutOut("num = %d\n", num);
    TutOut("grammar = %s\n", grammar);
    if (priority_known) {
        TutOut("priority = %d\n", priority);
    }
    else {
        TutOut("priority = unknown\n");
    }
    TutOut("delivery mode = %d\n", delivery_mode);
    TutOut("user prop = %d\n", user_prop);
    TutOut("num read cb = %d\n", num_read_cb);
    TutOut("num write cb = %d\n", num_write_cb);
    TutOut("num process cb = %d\n", num_process_cb);
    TutOut("num queue cb = %d\n", num_queue_cb);
    TutOut("total msg send = %d\n", total_msg_send);
    TutOut("total msg rcv = %d\n", total_msg_rcv);
    TutOut("total byte send = %d\n", total_byte_send);
    TutOut("total byte rcv = %d\n", total_byte_rcv);
}

/* make sure we reached the end of the message */
if (TutErrNumGet() != T_ERR_MSG_EOM) {
    return; /* error */
}
} /* process_mon_client_msg_type_poll_result */

```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientMsgTypePoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_MSG_TYPE_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
                           process_mon_client_msg_type_poll_result,
                           NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonClientNamesGetWatch

Name	TipcMonClientNamesGetWatch — determine if this RTclient is watching RTclient names
Synopsis	<pre>T_BOOL TipcMonClientNamesGetWatch(<i>watch_status_return</i>) T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the RTclient names watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonClientNamesGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>watch_status_return</i> was null
Description	TipcMonClientNamesGetWatch gets whether this RTclient is watching the RTclient names in the current project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_NAMES_STATUS message each time an RTclient is created or destroyed. An RTclient is considered created when it connects to RTserver. An RTclient is considered destroyed when it disconnects from RTserver. Each MON_CLIENT_NAMES_STATUS message contains five fields: • a STR_ARRAY field containing the current RTclient names • a STR_ARRAY field containing RTclient info strings of the form "<i>ident: user@node</i>" (such as "Runtime_RTie: ssuser@workstation1") • a STR_ARRAY field containing the name of the RTserver each RTclient is connected to • a STR field containing the name of the created RTclient (or "" if not applicable) • a STR field containing the name of the destroyed RTclient (or "" if not applicable) To set whether this RTclient is watching RTclient names, use TipcMonClientNamesSetWatch. To poll once for the RTclient names, use TipcMonClientNamesPoll.
Caution	If TipcMonClientNamesGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i> .
See Also	TipcMonClientNamesPoll, TipcMonClientNamesSetWatch

Examples This example uses TipcMonClientNamesGetWatch to get the status of watching RTclient names:

```
T_BOOL watch_status;

if (!TipcMonClientNamesGetWatch(&watch_status)) {
    return; /* error */
}
TutOut("This process is %s RTclient names.\n",
       watch_status ? "watching" : "not watching");
```

TipcMonClientNamesNumPoll

Name	TipcMonClientNamesNumPoll — poll for the number of RTclients in the current project that match the value of the Monitor_Scope option
Synopsis	T_BOOL TipcMonClientNamesNumPoll()
Arguments	None
Return Values	TRUE if the poll for the number of RTclients was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientNamesNumPoll fails, it returns FALSE and sets the global SmartSockets error number to: any error number from TipcSrvMsgWrite
Description	TipcMonClientNamesNumPoll polls for the number of RTclients in the current project that match the value of the Monitor_Scope option by sending a MON_CLIENT_NAMES_NUM_POLL_CALL to the RTserver. The RTserver responds by sending back a MON_CLIENT_NAMES_NUM_POLL_RESULT message. Each MON_CLIENT_NAMES_NUM_POLL_RESULT message contains one field: an INT4 field containing the number of RTclients in the project that match Monitor_Scope If there are no RTclients in the project that match the value of Monitor_Scope, the MON_CLIENT_NAMES_NUM_POLL_RESULT message returns 0 for the number of clients.
Caution	TipcMonClientNamesNumPoll sends a MON_CLIENT_NAMES_NUM_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering. The number of RTclients can only be polled, not watched.
See Also	TipcMonClientNamesPoll

Examples This example uses TipcMonClientNamesNumPoll to poll for the number of RTclient names, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT4 num_client_names;

/* send the poll request out to RTserver */
if (!TipcMonClientNamesNumPoll()) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_NAMES_NUM_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvMsgSearchType(10.0, mt);
if (msg == NULL) {
    return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}

if (!TipcMsgNextInt4(msg, &num_client_names)) {
    return; /* error */
}

TutOut("Got CLIENT_NAMES_NUM poll result.\n");
TutOut("number of client names = %d\n", num_client_names);

/* clean up */
if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcMonClientNamesPoll

Name	TipcMonClientNamesPoll — poll once for RTclient names
Synopsis	<code>T_BOOL TipcMonClientNamesPoll()</code>
Arguments	None
Return Values	TRUE if the poll for RTclient names was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientNamesPoll fails, it returns FALSE and sets the global SmartSockets error number to: • any error number from TipcSrvMsgWrite
Description	TipcMonClientNamesPoll polls for the RTclient names in the current project that match the value of the Monitor_Scope option by sending a MON_CLIENT_NAMES_POLL_CALL message to RTserver. RTserver responds by sending back a MON_CLIENT_NAMES_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. The MON_CLIENT_NAMES_POLL_RESULT message contains three fields: • a STR_ARRAY field containing the current RTclient names • a STR_ARRAY field containing RTclient info strings of the form "<i>ident: user@node</i>" (such as "Primary_RTclient: ssuser@workstation1") • a STR_ARRAY field containing the name of the RTserver each RTclient is connected to To watch for created and destroyed RTclient names, use TipcMonClientNamesSetWatch.
Caution	TipcMonClientNamesPoll sends a MON_CLIENT_NAMES_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.
See Also	TipcMonClientNamesGetWatch, TipcMonClientNamesSetWatch, TipcSrvMsgWrite

Examples This example uses TipcMonClientNamesPoll to poll for RTclient names, waits up to 10 seconds for the poll result, and then prints the poll result message:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *client_names;
T_INT4 num_client_names;
T_STR *info_strs;
T_STR *server_names;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcMonClientNamesPoll()) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_NAMES_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvMsgSearchType(10.0, mt);
if (msg == NULL) {
    return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}

/* client_names, info_strs, and server_names contains the same number of elements */
if (!TipcMsgNextStrArray(msg, &client_names, &num_client_names)
    || !TipcMsgNextStrArray(msg, &info_strs, &num_client_names)
    || !TipcMsgNextStrArray(msg, &server_names, &num_client_names))
{
    return; /* error */
}

TutOut("Got CLIENT_NAMES poll result.\n");
for (counter = 0; counter < num_client_names; counter++) {
    TutOut("client_names[%d] = %s\n", counter,
        client_names[counter]);
    TutOut("info_strs[%d] = %s\n", counter, info_strs[counter]);
    TutOut("server_names[%d] = %s\n", counter,
        server_names[counter]);
}

/* clean up */
if (!TipcMsgDestroy(msg)) {
    return; /* error */
}

```

TipcMonClientNamesSetWatch

Name	TipcMonClientNamesSetWatch — start or stop watching RTclient names
Synopsis	<pre>T_BOOL TipcMonClientNamesSetWatch(<i>watch_status</i>) T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the RTclient names watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonClientNamesSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonClientNamesSetWatch sets whether this RTclient is watching the RTclient names in the current project that match the value of the Monitor_Scope option. TipcMonClientNamesSetWatch saves the watch status and sends a MON_CLIENT_NAMES_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_CLIENT_NAMES_STATUS message each time an RTclient is created or destroyed. An RTclient is considered created when it connects to RTserver. An RTclient is considered destroyed when it disconnects from RTserver. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. The Srv_Client_Names_Min_Msgs option, described in the <i>TIBCO SmartSockets User's Guide</i>, controls the number of fields returned in the MON_CLIENT_NAMES_STATUS message. If Srv_Client_Names_Min_Msgs is set to FALSE, each MON_CLIENT_NAMES_STATUS message contains the following fields: • a STR_ARRAY field containing the current RTclient names • a STR_ARRAY field containing RTclient info strings of the form "<i>ident</i>: <i>user@node</i>" (such as "Primary_RTclient: ssuser@workstation1") • a STR_ARRAY field containing the name of the RTserver each RTclient is connected to • a STR field containing the name of the created RTclient (or "" if not applicable) • a STR field containing the name of the destroyed RTclient (or "" if not applicable)

- a STR field containing the reason for the client's disconnection (or "" if not applicable). This field is returned if the RTserver version is 6.8 or later. The possible disconnect reasons known at the time this document was written are:
 - **disconnect** - The client cleanly disconnected.
 - **failure detected** - A failure was detected while trying to communicate with the client (in most instances this means a keepalive failed due to network failure or a non-responsive client).
 - **max_buffer exceeded** - The client was disconnected by the server because its write buffer exceeded the size allowed by the "client_max_buffer" option.
 - **socket closed** - The client unexpectedly closed the socket (an EOF was detected). This would typically occur when a client terminates without first disconnecting (e.g. was CTRL-C'd).
 - **nak failure** - The client was disconnected by the group manager due to unrecoverable loss / NAK failure (disconnect_client_on_nak_failure was set to TRUE).
 - **disconnected by group** - The client was disconnected by the group connection server (RTgms). This message is used only when the specific reason for the disconnect is not known or is unavailable (which should only happen with a 6.8+ RTserver and a pre-6.8 RTgms).
 - **group connection lost** - The connection to the client's group server (RTgms) was lost, either due to network failure or because the RTgms is no longer running, or because the RTserver disconnected the RTgms.
 - **server connection lost** - The connection to the client's server was lost, either due to network failure or because the server is no longer running.
 - **server out of memory** - The server dropped the connection because it ran out of memory while trying to read from or write to the connection.
 - **gms out of memory** - The gms dropped the connection because it ran out of memory while trying to read from or write to the connection.
 - **unknown** - The connection was closed for an unknown (or unhandled) reason.

In addition to sending a MON_CLIENT_NAMES_STATUS message each time an RTclient is created or destroyed, RTserver also sends an initial MON_CLIENT_NAMES_STATUS message when the watch status is set to TRUE, so that the RTclient does not have to wait for the next RTclient name change to receive the first MON_CLIENT_NAMES_STATUS message. User's can limit the size of subsequent MON_CLIENT_NAMES_STATUS messages to only the name of the RTclient created or destroyed and the reason for the destroy by setting the RTserver option Srv_Client_Names_Min_Msgs to TRUE.

To determine if this RTclient is watching RTclient names, use `TipcMonClientNamesGetWatch`. To poll once for the RTclient names, use `TipcMonClientNamesPoll`.

Caution `TipcMonClientNamesSetWatch` sends a `MON_CLIENT_NAMES_SET_WATCH` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

See Also `TipcMonClientNamesGetWatch`, `TipcMonClientNamesPoll`

Examples This example uses `TipcMonClientNamesSetWatch` to start watching RTclient names and creates a callback to process the incoming `MON_CLIENT_NAMES_STATUS` messages:

```
/* ===== */
/*..process_mon_client_names_status -- process a MON_CLIENT_NAMES_STATUS message */
void T_ENTRY process_mon_client_names_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR *client_names;
    T_INT4 num_client_names;
    T_STR *info_strs;
    T_STR *server_names;
    T_STR created_client;
    T_STR destroyed_client;
    T_STR reason;
    T_INT4 counter;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
```

```

/* client_names, info_strs, and server_names contains the same
number of elements */

    if (!TipcMsgNextStrArray(data->msg, &client_names,
                             &num_client_names)
        || !TipcMsgNextStrArray(data->msg, &info_strs,
                                 &num_client_names)
        || !TipcMsgNextStrArray(data->msg, &server_names,
                                 &num_client_names)
        || !TipcMsgNextStr(data->msg, &created_client)
        || !TipcMsgNextStr(data->msg, &destroyed_client)) {
        return; /* error */
    }
TutOut("Got CLIENT_NAMES status.\n");
for (counter = 0; counter < num_client_names; counter++) {
    TutOut("client_names[%d] = %s\n", counter,
           client_names[counter]);
    TutOut("info_strs[%d] = %s\n", counter, info_strs[counter]);
    TutOut("server_names[%d] = %s\n", counter,
           server_names[counter]);
}
TutOut("created RTclient = %s\n", created_client);
TutOut("destroyed RTclient = %s\n", destroyed_client);
if (TipcMsgNextStr(data->msg, &reason)) {
    TutOut("reason for disconnect = %s\n", reason);
}
} /* process_mon_client_names_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcMonClientNamesSetWatch(TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_NAMES_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_mon_client_names_status,
                           NULL) == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonClientOptionPoll

Name	TipcMonClientOptionPoll — poll once for RTclient option information
Synopsis	<pre>T_BOOL TipcMonClientOptionPoll(<i>client_name</i>, <i>option_name</i>) T_STR <i>client_name</i>; T_STR <i>option_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll for option information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope) <i>option_name</i> — name of option to poll for information (use T_IPC_MON_ALL to indicate a poll of all options)
Return Values	TRUE if the poll for RTclient option information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientOptionPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> or <i>option_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientOptionPoll polls for RTclient option information by sending a MON_CLIENT_OPTION_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_OPTION_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. If <i>option_name</i> is T_IPC_MON_ALL, then all options in the polled RTclient are polled. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_CLIENT_OPTION_POLL_RESULT message contains one fixed field followed by one or more groups of five fields. The fixed field is a STR field containing the name of the RTclient. The group of five fields is: • a STR field containing the name of the option • an INT2 field containing the option type (really T_OPT_TYPE) • a STR field containing the value of the option, converted to a string like the setopt command displays

- an INT4 field containing whether the option is required to have a value (that is, it cannot be UNKNOWN)
- a STR_ARRAY field containing the legal values for the option; this array is empty if not applicable

There are multiple responses only if *client_name* is a wildcard or T_IPC_MON_ALL.

To poll once for RTserver option information, use TipcMonServerOptionPoll.

For an example of how MON_CLIENT_OPTION_POLL_RESULT messages can be used, see the RTmon GDI Client Information window.

Caution TipcMonClientOptionPoll sends a MON_CLIENT_OPTION_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client_name* or the option named *option_name* does not exist; the poll is silently dropped.

RTclient option information can only be polled, not watched.

See Also TipcMonServerOptionPoll, TipcSrvMsgWrite

Examples This example uses TipcMonClientOptionPoll to poll for all option information in all RTclients (including this one), creates a callback to process the incoming MON_CLIENT_OPTION_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_option_poll_result -- process a
MON_CLIENT_OPTION_POLL_RESULT message */
void T_ENTRY process_mon_client_option_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_STR option_name;
    T_INT2 type; /* really (T_OPT_TYPE) */
    T_STR value_str;
    T_INT4 required; /* really (T_BOOL) */
    T_STR *legal_vals;
    T_INT4 num_legal_vals; /* is zero if legal_vals N/A */
    T_INT4 counter; /* loop counter */

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }
}
```

```

/* get first fixed field */
if (!TipcMsgNextStr(data->msg, &client_name)) {
    return; /* error */
}
TutOut("Got CLIENT_OPTION poll result.\n");
TutOut("RTclient name = %s\n", client_name);
/* loop through all groups */
while (TipcMsgRead(data->msg,
                    T_IPC_FT_STR, &option_name,
                    T_IPC_FT_INT2, &type,
                    T_IPC_FT_STR, &value_str,
                    T_IPC_FT_INT4, &required,
                    T_IPC_FT_STR_ARRAY, &legal_vals,
                    &num_legal_vals,
                    NULL)) {
    TutOut("option name = %s\n", option_name);
    TutOut("type = %d\n", type);
    TutOut("value str = %s\n", value_str);
    TutOut("required = %s\n", required ? "TRUE" : "FALSE");
    if (num_legal_vals > 0) {
        TutOut("legal vals:\n");
        for (counter = 0; counter < num_legal_vals; counter++) {
            TutOut("  %s\n", legal_vals[counter]);
        }
    }
}

/* make sure we reached the end of the message */
if (TutErrNumGet() != T_ERR_MSG_EOM) {
    return; /* error */
}
} /* process_mon_client_option_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientOptionPoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}
mt = TipcMtLookupByNum(T_MT_MON_CLIENT_OPTION_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
                           process_mon_client_option_poll_result,
                           NULL) == NULL) {
    return; /* error */
}

```

```
/* read and process all poll results (as well as any other incoming  
messages) that arrive within the next 10 seconds */  
if (!TipcSrvMainLoop(10.0)) {  
    return; /* error */  
}
```

TipcMonClientSubjectExPoll

Name TipcMonClientSubjectExPoll — poll once for RTclient subject information



This new function supersedes TipcMonClientSubjectPoll for clients that use release 6.7 or later.

Synopsis `T_BOOL TipcMonClientSubjectExPoll(client_name, subject_name)`
 `T_STR client_name;`
 `T_STR subject_name;`

Arguments *client_name* — name of RTclient to poll for subject information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)

subject_name — name of subject to poll for information (use T_IPC_MON_ALL to indicate a poll of all subjects in the polled RTclient)

Return Values TRUE if the poll for RTclient subject information was successfully initiated, FALSE otherwise.

Diagnostics If TipcMonClientSubjectExPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *client_name* or *subject_name* was null
- any error number from TipcSrvMsgWrite

Description TipcMonClientSubjectExPoll polls for RTclient subject information by sending a MON_CLIENT_SUBJECT_EX_POLL_CALL message to the *client_name* RTclient. The *client_name* argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_SUBJECT_EX_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. If *subject_name* is T_IPC_MON_ALL, then all subjects ever used in the polled RTclient are polled. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response.

Each `MON_CLIENT_SUBJECT_EX_POLL_RESULT` message contains one fixed field followed by five array fields. The fixed field is a `STR` field containing the name of the `RTclient`. The array fields are:

- a `STR_ARRAY` field containing the name of the subject
- an `INT8_ARRAY` field containing the number of messages the polled `RTclient` connection has sent to each subject
- an `INT8_ARRAY` field containing the number of messages the polled `RTclient` connection has received for each subject
- an `INT8_ARRAY` field containing the number of bytes the polled `RTclient` connection has sent for each subject
- an `INT8_ARRAY` field containing the number of bytes the polled `RTclient` connection has received for each subject

There are multiple responses only if *client_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for global message traffic information, use `TipcMonClientMsgTrafficPoll`. To poll once for message traffic information by message type, use `TipcMonClientMsgTypeExPoll`.

For an example of how `MON_CLIENT_SUBJECT_EX_POLL_RESULT` messages can be used, see the `RTmon GDI Client Subject Message Traffic` window.

Caution `TipcMonClientSubjectExPoll` sends a `MON_CLIENT_SUBJECT_EX_POLL_CALL` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no response message if the `RTclient` named *client_name* or the subject named *subject_name* does not exist; the poll is silently dropped.

`RTclient` subject information can only be polled, not watched.

There is no way to poll for `RTserver` subject information.

See Also `TipcMonClientMsgTrafficPoll`, `TipcSrvMsgWrite`

Examples This example uses TipcMonClientSubjectExPoll to poll for all subject information in all RTclients (including this one), creates a callback to process the incoming MON_CLIENT_SUBJECT_POLL_EX_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_subject_ex_poll_result -- process a
MON_CLIENT_SUBJECT_EX_POLL_RESULT message */

void T_ENTRY process_mon_client_subject_ex_poll_result
(
    T_IPC_CONN conn,
    T_IPC_CONN_PROCESS_CB_DATA data,
    T_CB_ARG arg
)
{
    T_MSG    msg;
    T_STR    client_name;
    T_STR    *subject_name;
    T_INT4    num_subjects;
    T_INT8    *total_msg_send;
    T_INT4    num_msg_send;
    T_INT8    *total_msg_recv;
    T_INT4    num_msg_recv;
    T_INT8    *total_byte_send;
    T_INT4    num_byte_send;
    T_INT8    *total_byte_recv;
    T_INT4    num_byte_recv;
    T_INT4    counter;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    msg = data->msg;
    if (!TipcMsgNextStr(msg, &client_name)
        || !TipcMsgNextStrArray(msg, &subject_name, &num_subjects)
        || !TipcMsgNextInt8Array(msg, &total_msg_send, &num_msg_send)
        || !TipcMsgNextInt8Array(msg, &total_msg_recv, &num_msg_recv)
        || !TipcMsgNextInt8Array(msg, &total_byte_send, &num_byte_send)
        || !TipcMsgNextInt8Array(msg, &total_byte_recv, &num_byte_recv))
    {
        return; /* error */
    }
    TutOut("Got CLIENT_SUBJECT_EX poll result.\n");
    TutOut("RTclient name = %s\n", client_name);
}
```

```

/* loop through all groups */
for (counter=0; counter<num_subjects; counter++) {
    TutOut("subject name = %s\n", subject_name[counter]);
}
/*
* IMPORTANT: The format string for T_INT8 differs by platform.
* The T_INT8_SPEC utility macro supplies the correct format
* string for each platform.
*/
    TutOut("total msg send = " T_INT8_SPEC "\n",
        total_msg_send[counter]);
    TutOut("total msg recv = " T_INT8_SPEC "\n",
        total_msg_recv[counter]);
    TutOut("total byte send = " T_INT8_SPEC "\n",
        total_byte_send[counter]);
    TutOut("total byte recv = " T_INT8_SPEC "\n",
        total_byte_recv[counter]);
}

} /* process_mon_client_subject_poll_result */

/*===== */
/*...code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBJECT_EX_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}
if (TipcSrvProcessCbCreate(mt,
    process_mon_client_subject_ex_poll_result,
    NULL) == NULL) {
    return; /* error */
}

/* send the poll request out to RTserver */
if (!TipcMonClientSubjectExPoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

```

/* get first fixed field */
if (!TipcMsgNextStr(data->msg, &client_name)) {
return; /* error */
}
TutOut("Got CLIENT_SUBJECT poll result.\n");
TutOut("RTclient name = %s\n", client_name);

/* loop through all groups */
while (TipcMsgRead(data->msg,
                    T_IPC_FT_STR, &subject_name,
                    T_IPC_FT_INT4, &total_msg_send,
                    T_IPC_FT_INT4, &total_msg_recv,
                    T_IPC_FT_INT4, &total_byte_send,
                    T_IPC_FT_INT4, &total_byte_recv,
                    NULL)) {
    TutOut("subject name = %s\n", subject_name);
    TutOut("total msg send = %d\n", total_msg_send);
    TutOut("total msg recv = %d\n", total_msg_recv);
    TutOut("total byte send = %d\n", total_byte_send);
    TutOut("total byte recv = %d\n", total_byte_recv);
}

/* make sure we reached the end of the message */
if (TutErrNumGet() != T_ERR_MSG_EOM) {
    return; /* error */
}
} /* process_mon_client_subject_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientSubjectPoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}
mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBJECT_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}
if (TipcSrvProcessCbCreate(mt,
                           process_mon_client_subject_poll_result,
                           NULL) == NULL) {
    return; /* error */
}
}
/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}
}

```


TipcMonClientSubjectPoll

Name TipcMonClientSubjectPoll — poll once for RTclient subject information

Obsolete

The previous function, `TipcMonClientSubjectExPoll`, supersedes this function for all new development. Use this older function only when operating with clients prior to release 6.7. This poll can truncate the statistics data that it returns.

Synopsis `T_BOOL TipcMonClientSubjectPoll(client_name, subject_name)`
 `T_STR client_name;`
 `T_STR subject_name;`

Arguments *client_name* — name of RTclient to poll for subject information (wildcard names allowed, or use `T_IPC_MON_ALL` to indicate a poll of all RTclients matching the value of `Monitor_Scope`)

subject_name — name of subject to poll for information (use `T_IPC_MON_ALL` to indicate a poll of all subjects in the polled RTclient)

Return Values TRUE if the poll for RTclient subject information was successfully initiated, FALSE otherwise.

Diagnostics If `TipcMonClientSubjectPoll` fails, it returns FALSE and sets the global SmartSockets error number to one of:

- `T_ERR_NULL_PTR` — *client_name* or *subject_name* was null
- any error number from `TipcSrvMsgWrite`

Description `TipcMonClientSubjectPoll` polls for RTclient subject information by sending a `MON_CLIENT_SUBJECT_POLL_CALL` message to the *client_name* RTclient. The *client_name* argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or `T_IPC_MON_ALL` to poll all RTclients in the project that match the value of the `Monitor_Scope` option. The polled RTclient responds by sending back a `MON_CLIENT_SUBJECT_POLL_RESULT` message. The response may or may not come back quickly, depending on what the polled RTclient is doing. If *subject_name* is `T_IPC_MON_ALL`, and the option `monitor_level` is set to `all`, then all subjects ever used in the polled RTclient are polled. The polling RTclient can use functions such as `TipcSrvMsgSearchType` or `TipcSrvMainLoop` to get the response.

Each `MON_CLIENT_SUBJECT_POLL_RESULT` message contains one fixed field followed by one or more groups of five fields. The fixed field is a STR field containing the name of the RTclient. The group of five fields is:

- a STR field containing the name of the subject
- an INT4 field containing the number of messages the polled RTclient connection has sent to this subject
- an INT4 field containing the number of messages the polled RTclient connection has received for this subject
- an INT4 field containing the number of bytes the polled RTclient connection has sent for this subject
- an INT4 field containing the number of bytes the polled RTclient connection has received for this subject

There are multiple responses only if *client_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for global message traffic information, use `TipcMonClientMsgTrafficPoll`. To poll once for message traffic information by message type, use `TipcMonClientMsgTypePoll`.

For an example of how `MON_CLIENT_SUBJECT_POLL_RESULT` messages can be used, see the RTmon GDI Client Subject Message Traffic window.

Caution `TipcMonClientSubjectPoll` sends a `MON_CLIENT_SUBJECT_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTclient named *client_name* or the subject named *subject_name* does not exist; the poll is silently dropped.

RTclient subject information can only be polled, not watched.

There is no way to poll for RTserver subject information.

See Also `TipcMonClientMsgTrafficPoll`, `TipcSrvMsgWrite`

Examples This example uses TipcMonClientSubjectPoll to poll for all subject information in all RTclients (including this one), creates a callback to process the incoming MON_CLIENT_SUBJECT_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_subject_poll_result -- process a
MON_CLIENT_SUBJECT_POLL_RESULT message */
void T_ENTRY process_mon_client_subject_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_STR subject_name;
    T_INT4 total_msg_send;
    T_INT4 total_msg_rcv;
    T_INT4 total_byte_send;
    T_INT4 total_byte_rcv;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get first fixed field */
    if (!TipcMsgNextStr(data->msg, &client_name)) {
        return; /* error */
    }
    TutOut("Got CLIENT_SUBJECT poll result.\n");
    TutOut("RTclient name = %s\n", client_name);

    /* loop through all groups */
    while (TipcMsgRead(data->msg,
        T_IPC_FT_STR, &subject_name,
        T_IPC_FT_INT4, &total_msg_send,
        T_IPC_FT_INT4, &total_msg_rcv,
        T_IPC_FT_INT4, &total_byte_send,
        T_IPC_FT_INT4, &total_byte_rcv,
        NULL)) {
        TutOut("subject name = %s\n", subject_name);
        TutOut("total msg send = %d\n", total_msg_send);
        TutOut("total msg rcv = %d\n", total_msg_rcv);
        TutOut("total byte send = %d\n", total_byte_send);
        TutOut("total byte rcv = %d\n", total_byte_rcv);
    }

    /* make sure we reached the end of the message */
    if (TutErrNumGet() != T_ERR_MSG_EOM) {
        return; /* error */
    }
} /* process_mon_client_subject_poll_result */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientSubjectPoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}
mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBJECT_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}
if (TipcSrvProcessCbCreate(mt,
                           process_mon_client_subject_poll_result,
                           NULL) == NULL) {
    return; /* error */
}
/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonClientSubscribeGetWatch

Name	TipcMonClientSubscribeGetWatch — determine if this RTclient is watching the subjects that an RTclient is subscribing to
Synopsis	<pre>T_BOOL TipcMonClientSubscribeGetWatch(<i>client_name</i>, <i>watch_status_return</i>) T_STR <i>client_name</i>; T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope) <i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the RTclient subscribe watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonClientSubscribeGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>client_name</i> or <i>watch_status_return</i> was null
Description	TipcMonClientSubscribeGetWatch gets whether this RTclient is watching the subjects that the <i>client_name</i> RTclient is subscribing to. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_SUBSCRIBE_STATUS message each time the watched RTclient starts or stops subscribing to a subject. Each MON_CLIENT_SUBSCRIBE_STATUS message contains four fields: - a STR field containing the name of the RTclient - a STR_ARRAY field containing the names of the current subjects that the RTclient is subscribing to - a STR field containing the name of the subject that the RTclient has just started subscribing to (or "" if not applicable) - a STR field containing the name of the subject that the RTclient has just stopped subscribing to (or "" if not applicable) To set whether this RTclient is watching the subjects that an RTclient is subscribing to, use TipcMonClientSubscribeSetWatch. To poll once for the subjects that an RTclient is subscribing to, use TipcMonClientSubscribePoll.

For an example of how `MON_CLIENT_SUBSCRIBE_STATUS` messages can be used, see the RTmon GDI Watch Client Subjects window.

Caution If `TipcMonClientSubscribeGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

See Also `TipcMonClientSubscribePoll`, `TipcMonClientSubscribeSetWatch`

Examples This example uses `TipcMonClientSubscribeGetWatch` to get the status of watching subjects being subscribed to by the `primary` RTclient:

```
T_BOOL watch_status;

if (!TipcMonClientSubscribeGetWatch("primary", &watch_status)) {
    return; /* error */
}
TutOut("This process is %s subjects being subscribed to by ",
      watch_status ? "watching" : "not watching");
TutOut("primary.\n");
```

TipcMonClientSubscribeNumPoll

Name	TipcMonClientSubscribeNumPoll — poll for the number of subjects subscribed to by an RTclient
Synopsis	<pre>T_BOOL TipcMonClientSubscribeNumPoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)
Return Values	TRUE if the poll for the number of subjects was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientSubscribeNumPoll fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>client_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientSubscribeNumPoll polls for the number of subjects subscribed to by an RTclient by sending a MON_CLIENT_SUBSCRIBE_NUM_POLL_CALL message to the RTserver. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The RTserver responds by sending back a MON_CLIENT_SUBSCRIBE_NUM_POLL_RESULT message. The response should come back quickly. Each MON_CLIENT_SUBSCRIBE_NUM_POLL_RESULT message contains two fields: • a STR_ARRAY field containing the list of client names that match the <i>client_name</i> argument • an INT4_ARRAY field containing the number of subjects subscribed to by each RTclient

Caution There is no response message if the RTclient named *client_name* does not exist; the poll is silently dropped.

TipcMonClientSubscribeNumPoll sends a MON_CLIENT_SUBSCRIBE_NUM_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

The number of subjects subscribed to by an RTclient can only be polled, not watched.

See Also TipcMonClientSubscribePoll

Examples This example uses TipcMonClientSubscribeNumPoll to poll for the number of subjects that all RTclients are subscribing to, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *client_names;
T_INT4 num_client_names;
T_INT4 *num_subject_subscribes;
T_INT4 num_num_subject_subscribes;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcMonClientSubscribeNumPoll(T_IPC_MON_ALL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBSCRIBE_NUM_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvMsgSearchType(10.0, mt);
if (msg == NULL) {
    return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}

if (!TipcMsgNextStrArray(msg, &client_names, &num_client_names)
    || !TipcMsgNextInt4Array(msg, &num_subject_subscribes,
                            &num_num_subject_subscribes)) {
    return; /* error */
}
```



```
TutOut("Got CLIENT_SUBSCRIBE_NUM poll result.\n");
for (counter = 0; counter < num_client_names; counter++) {
    TutOut("client_names[%d] = %s\n", counter,
client_names[counter]);
    TutOut("num_subject_subscribes[%d] = %d\n", counter,
num_subject_subscribes[counter]);
}

/* clean up */
if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcMonClientSubscribePoll

Name	TipcMonClientSubscribePoll — poll once for the subjects that an RTclient is subscribing to
Synopsis	<pre>T_BOOL TipcMonClientSubscribePoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll for subjects being subscribed to (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)
Return Values	TRUE if the poll for subjects being subscribed to was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientSubscribePoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientSubscribePoll polls for subjects that the <i>client_name</i> RTclient is subscribing to by sending a MON_CLIENT_SUBSCRIBE_POLL_CALL message to RTserver. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. RTserver responds by sending back a MON_CLIENT_SUBSCRIBE_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. The MON_CLIENT_SUBSCRIBE_POLL_RESULT message contains one or more groups of two fields: • a STR field containing the name of the RTclient • a STR_ARRAY field containing the names of the current subjects that the RTclient is subscribing to This group of two fields is repeated multiple times only if <i>client_name</i> is a wildcard or T_IPC_MON_ALL. To watch for subjects that an RTclient is subscribing to, use TipcMonClientSubscribeSetWatch.

Caution TipcMonClientSubscribePoll sends a MON_CLIENT_SUBSCRIBE_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

See Also TipcMonClientSubscribeGetWatch, TipcMonClientSubscribeSetWatch, TipcSrvMsgWrite

Examples This example uses TipcMonClientSubscribePoll to poll for the subjects that all RTclients are subscribing to, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR client_name;
T_STR *subscribe_subject_names;
T_INT4 num_subscribe_subject_names;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcMonClientSubscribePoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBSCRIBE_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvMsgSearchType(10.0, mt);
if (msg == NULL) {
    return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}

/* loop through all groups */
TutOut("Got CLIENT_SUBSCRIBE poll result.\n");
while (TipcMsgNextStr(msg, &client_name)
      && TipcMsgNextStrArray(msg, &subscribe_subject_names,
                             &num_subscribe_subject_names)) {
    TutOut("RTclient name = %s\n", client_name);
    for (counter = 0; counter < num_subscribe_subject_names;
         counter++) {
        TutOut("subscribe_subject_names[%d] = %s\n", counter,
               subscribe_subject_names[counter]);
    }
}
```

```
/* make sure we reached the end of the message */
if (TutErrNumGet() != T_ERR_MSG_EOM) {
    return; /* error */
}

/* clean up */
if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcMonClientSubscribeSetWatch

Name	TipcMonClientSubscribeSetWatch — start or stop watching the subjects that an RTclient is subscribing to
Synopsis	<pre>T_BOOL TipcMonClientSubscribeSetWatch(<i>client_name</i>, <i>watch_status</i>) T_STR <i>client_name</i>; T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to set watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope) <i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the RTclient subscribe watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonClientSubscribeSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> was null • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonClientSubscribeSetWatch sets whether this RTclient is watching the subjects that the <i>client_name</i> RTclient is subscribing to. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. TipcMonClientSubscribeSetWatch saves the watch status and sends a MON_CLIENT_SUBSCRIBE_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_CLIENT_SUBSCRIBE_STATUS message each time the watched RTclient starts or stops subscribing to a subject. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.

Each `MON_CLIENT_SUBSCRIBE_STATUS` message contains four fields:

- a `STR` field containing the name of the `RTclient`
- a `STR_ARRAY` field containing the names of the current subjects that the `RTclient` is subscribing to
- a `STR` field containing the name of the subject that the `RTclient` has just started subscribing to (or "" if not applicable)
- a `STR` field containing the name of the subject that the `RTclient` has just stopped subscribing to (or "" if not applicable)

In addition to sending a `MON_CLIENT_SUBSCRIBE_STATUS` message each time the watched `RTclient` starts or stops subscribing to a subject, `RTserver` also sends an initial `MON_CLIENT_SUBSCRIBE_STATUS` message (or many messages if a wildcard or `T_IPC_MON_ALL` is used) when the watch status is set to `TRUE`, so that the `RTclient` does not have to wait for the next subscribed subject change to receive the first `MON_CLIENT_SUBSCRIBE_STATUS` message.

To determine if this `RTclient` is watching the subjects that an `RTclient` is subscribing to, use `TipcMonClientSubscribeGetWatch`. To poll once for the subjects that an `RTclient` is subscribing to, use `TipcMonClientSubscribePoll`.

For an example of how `MON_CLIENT_SUBSCRIBE_STATUS` messages can be used, see the `RTmon GDI Watch Client Subjects` window.

Caution `TipcMonClientSubscribeSetWatch` sends a `MON_CLIENT_SUBSCRIBE_SET_WATCH` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no initial status message if the `RTclient` named *client_name* does not exist.

See Also `TipcMonClientSubscribeGetWatch`, `TipcMonClientSubscribePoll`

Examples This example uses `TipcMonClientSubscribeSetWatch` to start watching the subjects being subscribed to by the primary `RTclient` and creates a callback to process the incoming `MON_CLIENT_SUBSCRIBE_STATUS` messages:

```
/* ===== */
/*..process_mon_client_subscribe_status -- process a MON_CLIENT_SUBSCRIBE_STATUS
message */
void T_ENTRY process_mon_client_subscribe_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_STR *subscribe_subject_names;
    T_INT4 num_subscribe_subject_names;
```

```

T_STR start_subject;
T_STR stop_subject;
T_INT4 counter;

/* must set current field first */
if (!TipcMsgSetCurrent(data->msg, 0)) {
    return; /* error */
}

/* get the fields from the message */
if (!TipcMsgNextStr(data->msg, &client_name)
    || !TipcMsgNextStrArray(data->msg, &subscribe_subject_names,
                            &num_subscribe_subject_names)
    || !TipcMsgNextStr(data->msg, &start_subject)
    || !TipcMsgNextStr(data->msg, &stop_subject)) {
    return; /* error */
}

TutOut("Got CLIENT_SUBSCRIBE status.\n");
TutOut("RTclient name = %s\n", client_name);
for (counter = 0; counter < num_subscribe_subject_names;
     counter++) {
    TutOut("subscribe_subject_names[%d] = %s\n", counter,
           subscribe_subject_names[counter]);
}
TutOut("start subject = %s\n", start_subject);
TutOut("stop subject = %s\n", stop_subject);
} /* process_mon_client_subscribe_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcMonClientSubscribeSetWatch("primary", TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBSCRIBE_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
                           process_mon_client_subscribe_status,
                           NULL) == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonClientTimeGetWatch

Name	TipcMonClientTimeGetWatch — determine if this RTclient is watching time information in an RTclient
Synopsis	<pre>T_BOOL TipcMonClientTimeGetWatch(<i>client_name</i>, <i>watch_status_return</i>) T_STR <i>client_name</i>; T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope) <i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the RTclient time information watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonClientTimeGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>client_name</i> or <i>watch_status_return</i> was null
Description	TipcMonClientTimeGetWatch gets whether this RTclient is watching time information in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_TIME_STATUS message each time the current time changes in the watched RTclient. Each MON_CLIENT_TIME_STATUS message contains seven fields: • a STR field containing the name of the RTclient • a REAL8 field containing the current time (the time from TutGetCurrentTime) • a STR field containing the current time converted to a string using the format specified in the Time_Format option; the time format from the watched RTclient, not this RTclient, is used • a REAL8 field containing the wall time (the time from TutGetWallTime) • a STR field containing the wall time converted to a string using the full time format • a REAL8 field containing the amount of CPU time used • an INT4 field containing the number of frames processed

To set whether this RTclient is watching RTclient time information, use `TipcMonClientTimeSetWatch`. To poll once for RTclient time information, use `TipcMonClientTimePoll`.

For an example of how `MON_CLIENT_TIME_STATUS` messages can be used, see the RTmon GDI Watch Client Time window.

Caution If `TipcMonClientTimeGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

See Also `TipcMonClientTimePoll`, `TipcMonClientTimeSetWatch`

Examples This example uses `TipcMonClientTimeGetWatch` to get the status of watching time information in the primary RTclient:

```
T_BOOL watch_status;

if (!TipcMonClientTimeGetWatch("primary", &watch_status)) {
    return; /* error */
}
TutOut("This process is %s time information in primary.\n",
       watch_status ? "watching" : "not watching");
```

TipcMonClientTimePoll

Name	TipcMonClientTimePoll — poll once for RTclient time information
Synopsis	<pre>T_BOOL TipcMonClientTimePoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll for time information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)
Return Values	TRUE if the poll for RTclient time information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientTimePoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientTimePoll polls for RTclient time information by sending a MON_CLIENT_TIME_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_TIME_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_CLIENT_TIME_POLL_RESULT message contains seven fields: • a STR field containing the name of the RTclient • a REAL8 field containing the current time (the time from TutGetCurrentTime) • a STR field containing the current time converted to a string using the format specified in the Time_Format option; the time format from the polled RTclient, not this RTclient, is used • a REAL8 field containing the wall time (the time from TutGetWallTime) • a STR field containing the wall time converted to a string using the full time format

- a REAL8 field containing the amount of CPU time used
- an INT4 field containing the number of frames processed

There are multiple responses only if *client_name* is a wildcard or T_IPC_MON_ALL.

To watch RTclient time information, use TipcMonClientTimeSetWatch.

Caution TipcMonClientTimePoll sends a MON_CLIENT_TIME_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client_name* does not exist; the poll is silently dropped.

See Also TipcMonClientTimeGetWatch, TipcMonClientTimeSetWatch, TipcSrvMsgWrite

Examples This example uses TipcMonClientTimePoll to poll for time information in all RTclients (including this one), creates a callback to process the incoming MON_CLIENT_TIME_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_time_poll_result -- process a MON_CLIENT_TIME_POLL_RESULT
message */
void T_ENTRY process_mon_client_time_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_REAL8 current_time;
    T_STR current_time_str;
    T_REAL8 wall_time;
    T_STR wall_time_str;
    T_REAL8 cpu_time;
    T_INT4 frame_count;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }
}
```

```

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg,
                     T_IPC_FT_STR, &client_name,
                     T_IPC_FT_REAL8, &current_time,
                     T_IPC_FT_STR, &current_time_str,
                     T_IPC_FT_REAL8, &wall_time,
                     T_IPC_FT_STR, &wall_time_str,
                     T_IPC_FT_REAL8, &cpu_time,
                     T_IPC_FT_INT4, &frame_count,
                     NULL)) {
        return; /* error */
    }

    TutOut("Got CLIENT_TIME poll result.\n");
    TutOut("RTclient name = %s\n", client_name);
    TutOut("current time = %f\n", current_time);
    TutOut("current time str = %s\n", current_time_str);
    TutOut("wall time = %f\n", wall_time);
    TutOut("wall time str = %s\n", wall_time_str);
    TutOut("CPU time = %f\n", cpu_time);
    TutOut("frame count = %d\n", frame_count);
} /* process_mon_client_time_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonClientTimePoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_TIME_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
process_mon_client_time_poll_result,
                           NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonClientTimeSetWatch

Name	TipcMonClientTimeSetWatch — start or stop watching RTclient time information
Synopsis	<pre>T_BOOL TipcMonClientTimeSetWatch(<i>client_name</i>, <i>watch_status</i>) T_STR <i>client_name</i>; T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to set watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope) <i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the RTclient time information watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonClientTimeSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: - T_ERR_NULL_PTR — <i>client_name</i> was null - T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE - T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonClientTimeSetWatch sets whether this RTclient is watching time information in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. TipcMonClientTimeSetWatch saves the watch status and sends a MON_CLIENT_TIME_SET_WATCH message to the <i>client_name</i> RTclient. The watch status controls whether the RTclient receives a MON_CLIENT_TIME_STATUS message each time the current time changes in the watched RTclient. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. Each MON_CLIENT_TIME_STATUS message contains seven fields: - a STR field containing the name of the RTclient - a REAL8 field containing the current time (the time from TutGetCurrentTime) - a STR field containing the current time converted to a string using the format specified in the Time_Format option; the time format from the watched RTclient, not this RTclient, is used - a REAL8 field containing the wall time (the time from TutGetWallTime)

- a STR field containing the wall time converted to a string using the full time format
- a REAL8 field containing the amount of CPU time used
- an INT4 field containing the number of frames processed

In addition to sending a MON_CLIENT_TIME_STATUS message each time the current time changes, the watched RTclient also sends an initial MON_CLIENT_TIME_STATUS message when the watch status is set to TRUE, so that the RTclient does not have to wait for the next time change to receive the first MON_CLIENT_TIME_STATUS message.

To determine if this RTclient is watching RTclient time information, use TipcMonClientTimeGetWatch. To poll once for RTclient time information, use TipcMonClientTimePoll.

For an example of how MON_CLIENT_TIME_STATUS messages can be used, see the RTmon GDI Watch Client Time window.

Caution TipcMonClientTimeSetWatch sends a MON_CLIENT_TIME_SET_WATCH message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no initial status message if the RTclient named *client_name* does not exist.

See Also TipcMonClientTimeGetWatch, TipcMonClientTimePoll

Examples This example uses TipcMonClientTimeSetWatch to start watching time information in the primary RTclient and creates a callback to process the incoming MON_CLIENT_TIME_STATUS messages:

```
/* ===== */
/*..process_mon_client_time_status -- process a MON_CLIENT_TIME_STATUS message */
void T_ENTRY process_mon_client_time_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_REAL8 current_time;
    T_STR current_time_str;
    T_REAL8 wall_time;
    T_STR wall_time_str;
    T_REAL8 cpu_time;
    T_INT4 frame_count;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }
}
```

```

/* get the fields from the message */
if (!TipcMsgRead(data->msg,
                 T_IPC_FT_STR, &client_name,
                 T_IPC_FT_REAL8, &current_time,
                 T_IPC_FT_STR, &current_time_str,
                 T_IPC_FT_REAL8, &wall_time,
                 T_IPC_FT_STR, &wall_time_str,
                 T_IPC_FT_REAL8, &cpu_time,
                 T_IPC_FT_INT4, &frame_count,
                 NULL)) {
    return; /* error */
}

TutOut("Got CLIENT_TIME status.\n");
TutOut("RTclient name = %s\n", client_name);
TutOut("current time = %f\n", current_time);
TutOut("current time str = %s\n", current_time_str);
TutOut("wall time = %f\n", wall_time);
TutOut("wall time str = %s\n", wall_time_str);
TutOut("CPU time = %f\n", cpu_time);
TutOut("frame count = %d\n", frame_count);
} /* process_mon_client_time_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcMonClientTimeSetWatch("primary", TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_TIME_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_mon_client_time_status,
NULL)
    == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonClientVersionPoll

Name	TipcMonClientVersionPoll — poll once for RTclient version information
Synopsis	<pre>T_BOOL TipcMonClientVersionPoll(<i>client_name</i>) T_STR <i>client_name</i>;</pre>
Arguments	<i>client_name</i> — name of RTclient to poll for version information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)
Return Values	TRUE if the poll was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonClientVersionPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>client_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonClientVersionPoll polls for RTclient version information by sending a MON_CLIENT_VERSION_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclients respond by sending back a MON_CLIENT_VERSION_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_CLIENT_VERSION_POLL_RESULT message contains two fields: • a STR field containing the name of the RTclient • a INT4 field containing the version information of the RTclient There are multiple responses only if <i>client_name</i> is a wildcard or T_IPC_MON_ALL.
Caution	TipcMonClientVersionPoll sends a MON_CLIENT_VERSION_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering. There is no response message if the RTclient named <i>client_name</i> does not exist; the poll is silently dropped. RTclient version information can be polled, but it cannot be watched.

See Also TipcMonServerVersionPoll

Examples This example uses TipcMonClientVersionPoll to poll for version information in all RTclients (including this one), creates a callback to process the incoming MON_CLIENT_VERSION_POLL_RESULT messages, and then waits up to 10 seconds for the poll results.

```
void T_ENTRY ClientVersionCb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR client_name;
    T_INT4 version;

    TipcMsgNextStr(data->msg, &client_name);
    TipcMsgNextInt4(data->msg, &version);
    TutOut("%s: %d\n", client_name, version);
} /* ClientVersionCb */

/* ... code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_VERSION_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, ClientVersionCb, NULL) == NULL) {
    return; /* error */
}

TipcMonClientVersionPoll("/...");
TipcSrvFlush();
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}
```

TipcMonExtDelete

Name	TipcMonExtDelete — remove a field from an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtDelete(<i>field_name</i>) T_STR <i>field_name</i>;</pre>
Arguments	<i>field_name</i> — name of the field to remove from the RTclient’s MON_CLIENT_EXT_POLL_RESULT message.
Return Values	TRUE if the field was removed, FALSE otherwise.
Diagnostics	If TipcMonExtDelete fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgDeleteNamedField
Description	TipcMonExtDelete removes a field from the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. The effected MON_CLIENT_EXT_POLL_RESULT message is the one belonging to the RTclient who issued TipcMonExtDelete.
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetBinary

Name	TipcMonExtSetBinary — add or update a field of type BINARY in an RTclient's MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetBinary(<i>field_name</i>, <i>data</i>, <i>size</i>) T_STR <i>field_name</i>; T_PTR <i>data</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. <i>size</i> — number of bytes in the field.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetBinary fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedBinary • any error number from TipcMsgUpdateNamedBinary
Description	TipcMonExtSetBinary creates a field of type BINARY in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetBinary only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetBool

Name	TipcMonExtSetBool — add or update a field of type BOOL in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetBool(<i>field_name</i>, <i>data</i>) T_STR <i>field_name</i>; T_BOOL <i>data</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetBool fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedBool • any error number from TipcMsgUpdateNamedBool
Description	TipcMonExtSetBool creates a field of type BOOL in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetBool only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetBoolArray

Name	TipcMonExtSetBoolArray — add or update a field containing an array of type BOOL in an RTclient's MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetBoolArray(<i>field_name</i>, <i>data</i>, <i>size</i>) T_STR <i>field_name</i>; T_BOOL *<i>data</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. <i>size</i> — number of elements in the field.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetBoolArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedBoolArray • any error number from TipcMsgUpdateNamedBoolArray
Description	TipcMonExtSetBoolArray creates a field containing an array of type BOOL in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetBoolArray only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetInt2

Name	TipcMonExtSetInt2 — add or update a field of type INT2 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetInt2(<i>field_name</i>, <i>data</i>) T_STR <i>field_name</i>; T_INT2 <i>data</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetInt2 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedInt2 • any error number from TipcMsgUpdateNamedInt2
Description	TipcMonExtSetInt2 creates a field of type INT2 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetInt2 only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetInt2Array

Name	TipcMonExtSetInt2Array — add or update a field containing an array of type INT2 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetInt2Array(<i>field_name</i>, <i>data</i>, <i>size</i>) T_STR <i>field_name</i>; T_INT2 *<i>data</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. <i>size</i> — number of elements in the field.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetInt2Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedInt2Array • any error number from TipcMsgUpdateNamedInt2Array
Description	TipcMonExtSetInt2Array creates a field containing an array of type INT2 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetInt2Array only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetInt4

Name	TipcMonExtSetInt4 — add or update a field of type INT4 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetInt4(<i>field_name</i>, <i>data</i>) T_STR <i>field_name</i>; T_INT4 <i>data</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetInt4 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedInt4 • any error number from TipcMsgUpdateNamedInt4
Description	TipcMonExtSetInt4 creates a field of type INT4 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetInt4 only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetInt4Array

Name	TipcMonExtSetInt4Array — add or update a field containing an array of type INT4 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetInt4Array(<i>field_name</i>, <i>data</i>, <i>size</i>) T_STR <i>field_name</i>; T_INT4 *<i>data</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. <i>size</i> — number of elements in the field.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetInt4Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedInt4Array • any error number from TipcMsgUpdateNamedInt4Array
Description	TipcMonExtSetInt4Array creates a field containing an array of type INT4 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetInt4Array only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetInt8

Name	TipcMonExtSetInt8 — add or update a field of type INT8 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetInt8(<i>field_name</i>, <i>data</i>) T_STR <i>field_name</i>; T_INT8 <i>data</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetInt8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedInt8 • any error number from TipcMsgUpdateNamedInt8
Description	TipcMonExtSetInt8 creates a field of type INT8 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetInt8 only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetInt8Array

Name	TipcMonExtSetInt8Array — add or update a field containing an array of type INT8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetInt8Array(<i>field_name</i>, <i>data</i>, <i>size</i>) T_STR <i>field_name</i>; T_INT8 *<i>data</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. <i>size</i> — number of elements in the field.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetInt8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedInt8Array • any error number from TipcMsgUpdateNamedInt8Array
Description	TipcMonExtSetInt8Array creates a field containing an array of type INT8 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetInt8Array only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetReal4

Name	TipcMonExtSetReal4 — add or update a field of type REAL4 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetReal4(<i>field_name</i>, <i>data</i>) T_STR <i>field_name</i>; T_REAL4 <i>data</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetReal4 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedReal4 • any error number from TipcMsgUpdateNamedReal4
Description	TipcMonExtSetReal4 creates a field of type REAL4 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetReal4 only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetReal4Array

Name	TipcMonExtSetReal4Array — add or update a field containing an array of type REAL4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetReal4Array(<i>field_name</i>, <i>data</i>, <i>size</i>) T_STR <i>field_name</i>; T_REAL4 *<i>data</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. <i>size</i> — number of elements in the field.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetReal4Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedReal4Array • any error number from TipcMsgUpdateNamedReal4Array
Description	TipcMonExtSetReal4Array creates a field containing an array of type REAL4 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetReal4Array only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetReal8

Name	TipcMonExtSetReal8 — adds or updates a field of type REAL8 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetReal8(<i>field_name</i>, <i>data</i>) T_STR <i>field_name</i>; T_REAL8 <i>data</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetReal8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedReal8 • any error number from TipcMsgUpdateNamedReal8
Description	TipcMonExtSetReal8 creates a field of type REAL8 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetReal8 only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetReal8Array

Name	TipcMonExtSetReal8Array — add or update a field containing an array of type REAL8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetReal8Array(<i>field_name</i>, <i>data</i>, <i>size</i>) T_STR <i>field_name</i>; T_REAL8 *<i>data</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. <i>size</i> — number of elements in the field.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetReal8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedReal8Array • any error number from TipcMsgUpdateNamedReal8Array
Description	TipcMonExtSetReal8Array creates a field containing an array of type REAL8 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetReal8Array only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetReal16

Name	TipcMonExtSetReal16 — add or update a field of type REAL16 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetReal16(<i>field_name</i>, <i>data</i>) T_STR <i>field_name</i>; T_REAL16 <i>data</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetReal16 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedReal16 • any error number from TipcMsgUpdateNamedReal16
Description	TipcMonExtSetReal16 creates a field of type REAL16 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetReal16 only updates it with the value of <i>data</i> .
	Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetReal16Array

Name	TipcMonExtReal16Array — add or update a field containing an array of type REAL16 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetReal16Array(<i>field_name</i>, <i>data</i>, <i>size</i>) T_STR <i>field_name</i>; T_REAL16 *<i>data</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. <i>size</i> — number of elements in the field.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetReal16Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedReal16Array • any error number from TipcMsgUpdateNamedReal16Array
Description	TipcMonExtSetReal16Array creates a field containing an array of type REAL16 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetReal16Array only updates it with the value of <i>data</i> .
	Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetStr

Name	TipcMonExtSetStr — add or update a field of type STR in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetStr(<i>field_name</i>, <i>data</i>) T_STR <i>field_name</i>; T_STR <i>data</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetStr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedStr • any error number from TipcMsgUpdateNamedStr
Description	TipcMonExtSetStr creates a field of type STR in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetStr only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetStrArray

Name	TipcMonExtSetStrArray — add or update a field containing an array of type STR in an RTclient's MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetStrArray(<i>field_name</i>, <i>data</i>, <i>size</i>) T_STR <i>field_name</i>; T_STR *<i>data</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. <i>size</i> — number of elements in the field.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetStrArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedStrArray • any error number from TipcMsgUpdateNamedStrArray
Description	TipcMonExtSetStrArray creates a field containing an array of type STR in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetStrArray only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetUtf8

Name	TipcMonExtSetUtf8 — add or update a field of type UTF8 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetUtf8(<i>field_name</i>, <i>data</i>) T_STR <i>field_name</i>; T_STR <i>data</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetUtf8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedUtf8 • any error number from TipcMsgUpdateNamedUtf8
Description	TipcMonExtSetUtf8 creates a field of type UTF8 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetUtf8 only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonExtSetUtf8Array

Name	TipcMonExtSetUtf8Array — add or update a field containing an array of type UTF8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message
Synopsis	<pre>T_BOOL TipcMonExtSetUtf8Array(<i>field_name</i>, <i>data</i>, <i>size</i>) T_STR <i>field_name</i>; T_STR *<i>data</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message. <i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. <i>size</i> — number of elements in the field.
Return Values	TRUE if the field was added or updated, FALSE otherwise.
Diagnostics	If TipcMonExtSetUtf8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field_name</i> was null • any error number from TipcMsgAddNamedUtf8Array • any error number from TipcMsgUpdateNamedUtf8Array
Description	TipcMonExtSetUtf8Array creates a field containing an array of type UTF8 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcMonExtSetUtf8Array only updates it with the value of <i>data</i> .
See Also	Use other TipcMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcMonClientExtPoll.
Examples	See the example for TipcMonClientExtPoll on page 213.

TipcMonGetIdentStr

Name	TipcMonGetIdentStr — get the monitoring identification string of this process
Synopsis	<pre>T_BOOL TipcMonGetIdentStr(<i>ident_str_return</i>) T_STR *<i>ident_str_return</i>;</pre>
Arguments	<i>ident_str_return</i> — storage for identification string
Return Values	TRUE if the identification string was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonGetIdentStr fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>ident_str_return</i> was null
Description	TipcMonGetIdentStr gets the monitoring identification string of this process. This identification string is used as a descriptive name for the process when it is being monitored. It is used as part of a field in these message types: • MON_CLIENT_NAMES_STATUS • MON_SERVER_NAMES_STATUS • MON_CLIENT_NAMES_POLL_RESULT • MON_SERVER_NAMES_POLL_RESULT In these message types, the full field has the form "<i>ident</i>: <i>user@node</i>" (for example, "Primary_RTclient: ssuser@workstation1". The monitoring identification string is used as <i>ident</i>. By default the monitoring identification string is based on the most descriptive counted license the process has (for example, Primary_RTclient is more descriptive than RTclient). This string can be set with the function TipcMonSetIdentStr.
Caution	The storage pointed to by <i>ident_str_return</i> should not be modified or freed. The storage is thread safe, but may be overwritten by subsequent calls to SmartSockets APIs within the same thread. If you wish to preserve the value, you should make a copy of the value.
See Also	TipcMonSetIdentStr

Examples This example gets and prints the monitoring identification string for this process:

```
T_STR ident_str;

if (!TipcMonGetIdentStr(&ident_str)) {
    return; /* error */
}
TutOut("Monitoring identification string is %s\n", ident_str);
```

TipcMonPrintWatch

Name	TipcMonPrintWatch — print all monitoring categories being watched
Synopsis	<pre>T_BOOL TipcMonPrintWatch(func) T_OUT_FUNC func;</pre>
Arguments	<i>func</i> — TutOut-style output function
Return Values	TRUE if the watch information was successfully printed, FALSE otherwise.
Diagnostics	If TipcMonPrintWatch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>func</i> was null
Description	TipcMonPrintWatch prints all the monitoring categories being watched to a TutOut-style output function. The output function is called once for each line of output.
Caution	None
Examples	This example starts watching a few monitoring categories and then calls TipcMonPrintWatch to print out what is being watched: <pre>if (!TipcMonSubjectSubscribeSetWatch(T_IPC_MON_ALL, TRUE) { return; /* error */ } if (!TipcMonProjectNamesSetWatch(TRUE) { return; /* error */ } if (!TipcMonClientTimeSetWatch("primary", TRUE) { return; /* error */ } if (!TipcMonClientMsgSubscribeSetWatch("primary", "numeric_data") { return; /* error */ } if (!TipcMonPrintWatch(TutOut)) { return; /* error */ }</pre> The above fragment would print this output: <pre>Watching project_names. Watching subject_subscribe <@>. Watching client_time <primary>. Watching client_msg_recv <primary> <numeric_data>.</pre>

TipcMonProjectNamesGetWatch

Name	TipcMonProjectNamesGetWatch — determine if this RTclient is watching project names
Synopsis	<pre>T_BOOL TipcMonProjectNamesGetWatch(<i>watch_status_return</i>) T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the project names watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonProjectNamesGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>watch_status_return</i> was null
Description	TipcMonProjectNamesGetWatch gets whether this RTclient is watching project names. The watch status controls whether the RTclient receives a MON_PROJECT_NAMES_STATUS message each time a project is created or destroyed. A project is considered created when the first RTclient in that project connects to RTserver. A project is considered destroyed when the last RTclient in that project disconnects from RTserver. Each MON_PROJECT_NAMES_STATUS message contains three fields: - a STR_ARRAY field containing the current project names - a STR field containing the name of the created project (or "" if not applicable) - a STR field containing the name of the destroyed project (or "" if not applicable) To set whether this RTclient is watching project names, use TipcMonProjectNamesSetWatch. To poll once for project names, use TipcMonProjectNamesPoll. For an example of how MON_PROJECT_NAMES_STATUS messages can be used, see the RTmon GDI Project Browser window.
Caution	If TipcMonProjectNamesGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i> .
See Also	TipcMonProjectNamesPoll, TipcMonProjectNamesSetWatch

Examples This example uses TipcMonProjectNamesGetWatch to get the status of watching project names:

```
T_BOOL watch_status;

if (!TipcMonProjectNamesGetWatch(&watch_status)) {
    return; /* error */
}
TutOut("This RTclient is %s project names.\n",
       watch_status ? "watching" : "not watching");
```

TipcMonProjectNamesPoll

Name	TipcMonProjectNamesPoll — poll once for project names
Synopsis	<code>T_BOOL TipcMonProjectNamesPoll()</code>
Arguments	None
Return Values	TRUE if the poll for project names was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonProjectNamesPoll fails, it returns FALSE and sets the global SmartSockets error number to: any error number from TipcSrvMsgWrite
Description	TipcMonProjectNamesPoll polls for project names by sending a MON_PROJECT_NAMES_POLL_CALL message to RTserver. RTserver responds by sending back a MON_PROJECT_NAMES_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. The MON_PROJECT_NAMES_POLL_RESULT message contains one field, a STR_ARRAY field containing the current project names. To watch for created and destroyed project names, use TipcMonProjectNamesSetWatch.
Caution	TipcMonProjectNamesPoll sends a MON_PROJECT_NAMES_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.
See Also	TipcMonProjectNamesGetWatch, TipcMonProjectNamesSetWatch, TipcSrvMsgWrite
Examples	This example uses TipcMonProjectNamesPoll to poll for project names, waits up to 10 seconds for the poll result, and then prints the poll result message: <pre>T_IPC_MT mt; T_IPC_MSG msg; T_STR *project_names; T_INT4 num_project_names; T_INT4 counter;</pre>

```

/* send the poll request out to RTserver */
if (!TipcMonProjectNamesPoll()) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_PROJECT_NAMES_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvMsgSearchType(10.0, mt);
if (msg == NULL) {
    return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}

if (!TipcMsgNextStrArray(msg, &project_names, &num_project_names))
{
    return; /* error */
}

TutOut("Got PROJECT_NAMES poll result.\n");
for (counter = 0; counter < num_project_names; counter++) {
    TutOut("project_names[%d] = %s\n", counter,
project_names[counter]);
}

/* clean up */
if (!TipcMsgDestroy(msg)) {
    return; /* error */
}

```

TipcMonProjectNamesSetWatch

Name	TipcMonProjectNamesSetWatch — start or stop watching project names
Synopsis	<pre>T_BOOL TipcMonProjectNamesSetWatch(<i>watch_status</i>) T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the project names watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonProjectNamesSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonProjectNamesSetWatch sets whether this RTclient is watching project names. TipcMonProjectNamesSetWatch saves the watch status and sends a MON_PROJECT_NAMES_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_PROJECT_NAMES_STATUS message each time a project is created or destroyed. A project is considered created when the first RTclient in that project connects to RTserver. A project is considered destroyed when the last RTclient in that project disconnects from RTserver. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. Each MON_PROJECT_NAMES_STATUS message contains three fields: • a STR_ARRAY field containing the current project names • a STR field containing the name of the created project (or "" if not applicable) • a STR field containing the name of the destroyed project (or "" if not applicable) In addition to sending a MON_PROJECT_NAMES_STATUS message each time a project is created or destroyed, RTserver also sends an initial MON_PROJECT_NAMES_STATUS message when the watch status is set to TRUE, so that the RTclient does not have to wait for the next project name change to receive the first MON_PROJECT_NAMES_STATUS message. To determine if this RTclient is watching project names, use TipcMonProjectNamesGetWatch. To poll once for project names, use TipcMonProjectNamesPoll.

For an example of how MON_PROJECT_NAMES_STATUS messages can be used, see the RTmon GDI Project Browser window.

Caution TipcMonProjectNamesSetWatch sends a MON_PROJECT_NAMES_SET_WATCH message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

See Also TipcMonProjectNamesGetWatch, TipcMonProjectNamesPoll

Examples This example uses TipcMonProjectNamesSetWatch to start watching project names and creates a callback to process the incoming MON_PROJECT_NAMES_STATUS messages:

```
/* ===== */
/*..process_mon_project_names_status -- process a MON_PROJECT_NAMES_STATUS message
*/
void T_ENTRY process_mon_project_names_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR *project_names;
    T_INT4 num_project_names;
    T_STR created_project;
    T_STR destroyed_project;
    T_INT4 counter;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgNextStrArray(data->msg, &project_names,
                            &num_project_names)
        || !TipcMsgNextStr(data->msg, &created_project)
        || !TipcMsgNextStr(data->msg, &destroyed_project)) {
        return; /* error */
    }

    TutOut("Got PROJECT_NAMES status.\n");
    for (counter = 0; counter < num_project_names; counter++) {
        TutOut("project_names[%d] = %s\n", counter,
              project_names[counter]);
    }
    TutOut("created project = %s\n", created_project);
    TutOut("destroyed project = %s\n", destroyed_project);
} /* process_mon_project_names_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcMonProjectNamesSetWatch(TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_PROJECT_NAMES_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_mon_project_names_status,
NULL)
    == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonServerBufferPoll

Name	TipcMonServerBufferPoll — poll once for RTserver message-related buffer information
Synopsis	<pre>T_BOOL TipcMonServerBufferPoll(<i>server_name</i>, <i>connected_process_name</i>) T_STR <i>server_name</i>; T_STR <i>connected_process_name</i>;</pre>
Arguments	<i>server_name</i> — name of RTserver to poll for buffer information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope) <i>connected_process_name</i> — name of connected process whose buffer in RTserver to poll (use T_IPC_MON_ALL to indicate a poll of all processes that the polled RTserver is connected to)
Return Values	TRUE if the poll for RTserver buffer information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonServerBufferPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>server_name</i> or <i>connected_process_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonServerBufferPoll polls for RTserver message-related buffer information by sending a MON_SERVER_BUFFER_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The buffer information of <i>connected_process_name</i> (which can be either an RTserver or another RTclient in the same project) is polled. The polled RTserver responds by sending back a MON_SERVER_BUFFER_POLL_RESULT message. The response should come back quickly. If <i>connected_process_name</i> is T_IPC_MON_ALL then the buffers in the polled RTserver of all connected processes are polled. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response.

Each `MON_SERVER_BUFFER_POLL_RESULT` message contains one fixed field followed by one or more groups of five fields. The fixed field is a STR field containing the name of the RTserver. The group of five fields is:

- a STR field containing the name of the process RTserver is connected to
- an INT4 field containing the number of queued incoming message in the RTserver's connection to the connected process
- an INT4 field containing the number of bytes of data in the queued incoming messages in the RTserver's connection to the connected process
- an INT4 field containing the number of bytes of data in the read buffer in the RTserver's connection to the connected process
- an INT4 field containing the number of bytes of data in the write buffer in the RTserver's connection to the connected process

There are multiple responses only if *server_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTclient buffer information, use `TipcMonClientBufferPoll`.

For an example of how `MON_SERVER_BUFFER_POLL_RESULT` messages can be used, see the RTmon GDI Server Buffers window.

Caution `TipcMonServerBufferPoll` sends a `MON_SERVER_BUFFER_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTserver named *server_name* or connected process named *connected_process_name* does not exist; the poll is silently dropped.

RTserver buffer information can only be polled, not watched.

See Also `TipcMonClientBufferPoll`, `TipcSrvMsgWrite`

Examples This example uses TipcMonServerBufferPoll to poll for all buffer information in all RTservers, creates a callback to process the incoming MON_SERVER_BUFFER_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_buffer_poll_result -- process a
MON_SERVER_BUFFER_POLL_RESULT message */
void T_ENTRY process_mon_server_buffer_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR server_name;
    T_STR connected_process_name;
    T_INT4 msg_queue_count;
    T_INT4 msg_queue_byte_count;
    T_INT4 read_buffer_count;
    T_INT4 write_buffer_count;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get first fixed field */
    if (!TipcMsgNextStr(data->msg, &server_name)) {
        return; /* error */
    }
    TutOut("Got SERVER_BUFFER poll result.\n");
    TutOut("RTserver name = %s\n", server_name);

    /* loop through all groups */
    while (TipcMsgRead(data->msg,
        T_IPC_FT_STR, &connected_process_name,
        T_IPC_FT_INT4, &msg_queue_count,
        T_IPC_FT_INT4, &msg_queue_byte_count,
        T_IPC_FT_INT4, &read_buffer_count,
        T_IPC_FT_INT4, &write_buffer_count,
        NULL)) {
        TutOut("connected process name = %s\n",
connected_process_name);
        TutOut("msg queue count = %d\n", msg_queue_count);
        TutOut("msg queue byte count = %d\n", msg_queue_byte_count);
        TutOut("read buffer count = %d\n", read_buffer_count);
        TutOut("write buffer count = %d\n", write_buffer_count);
    }

    /* make sure we reached the end of the message */
    if (TutErrNumGet() != T_ERR_MSG_EOM) {
        return; /* error */
    }
}
```

```

} /* process_mon_server_buffer_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonServerBufferPoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_BUFFER_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
    process_mon_server_buffer_poll_result,
    NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonServerCongestionGetWatch

Name	TipcMonServerCongestionGetWatch — determine if the RTserver is watching for congestion in the RTserver write buffer of a connected process
Synopsis	<pre>T_BOOL TipcMonServerCongestionGetWatch(server_name, connected_process_name, watch_status_return) T_STR server_name; T_STR connected_process_name; T_BOOL *watch_status_return;</pre>
Arguments	<i>server_name</i> — name of RTserver to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope) <i>connected_process_name</i> — the name of the process to get watch status of (use T_IPC_MON_ALL to indicate a poll of all processes that the polled RTserver is connected to) <i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonServerCongestionGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>server_name</i>, <i>connected_process_name</i>, or <i>watch_status_return</i> was null
Description	TipcMonServerCongestionGetWatch determines whether the <i>server_name</i> RTserver is watching for congestion in the <i>connected_process_name</i> write buffer. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The <i>connected_process_name</i> argument can be the name of any RTclient connected to <i>server_name</i>, or T_IPC_MON_ALL to indicate a poll of all processes that the polled RTserver is connected to. The <i>watch_status_return</i> argument is set to TRUE if the watch is currently active, FALSE if not.
Caution	TipcMonServerCongestionGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i>. Write buffer congestion can only be watched, not polled.

See Also TipcMonServerCongestionSetWatch, TipcMonClientCongestionSetWatch, TipcMonClientCongestionGetWatch

Examples This example uses TipcMonServerCongestionGetWatch to determine the status of a buffer congestion watch in all connections on all RTservers:

```
T_BOOL watch_status;

if (!TipcMonServerCongestionGetWatch("/...", T_IPC_MON_ALL,
                                     &watch_status)) {
    return; /* error */
}

TutOut("This RTclient is %s RTserver buffer congestion in ",
       watch_status ? "watching" : "not watching");
TutOut("all connections on all servers.\n");
```

TipcMonServerCongestionSetWatch

Name TipcMonServerCongestionSetWatch — start or stop watching for congestion in the RTserver write buffer of a connected process

Synopsis `T_BOOL TipcMonServerCongestionSetWatch(server_name, connected_process_name, high_water, low_water, watch_status)`
 `T_STR server_name;`
 `T_STR connected_process_name;`
 `T_INT4 high_water;`
 `T_INT4 low_water;`
 `T_BOOL watch_status;`

Arguments *server_name* — name of RTserver to set watch status (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)

connected_process_name — the name of the process whose write buffer status you want to start or stop watching (use T_IPC_MON_ALL to indicate a poll of all processes that the polled RTserver is connected to)

high_water — the number of pending bytes that must be in a buffer for it to be considered congested

low_water — the quantity that the number of pending bytes must fall below to end buffer congestion

watch_status — new watch status (TRUE to start watching, or FALSE to stop watching)

Return Values TRUE if the watch status was successfully set, FALSE otherwise.

Diagnostics If TipcMonServerCongestionSetWatch fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *server_name* was null
- T_ERR_VAL_INVALID — *watch_status* was not TRUE or FALSE, or the value of *low_water* was less than 0 or greater than the value of *high_water*
- T_ERR_VAL_SAME — *watch_status* is the same as the current watch status

Description TipcMonServerCongestionSetWatch starts or stops the watch of the *connected_process_name* write buffer in the *server_name* RTserver. The *connected_process_name* argument can be the name of any RTclient connected to *server_name*, or T_IPC_MON_ALL to indicate a poll of all processes that the polled

RTserver is connected to. The *server_name* argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option.

TIPCMonServerCongestionSetWatch saves the watch status and sends a MON_SERVER_CONGESTION_SET_WATCH message to the *server_name* RTserver. The watch status controls whether the RTclient receives a MON_SERVER_CONGESTION_STATUS message when the number of bytes in the process's write buffer passes the *high_water* or *low_water* threshold. When the write buffer reaches *high_water* messages pending, the RTserver sends a MON_SERVER_CONGESTION_STATUS message. The next MON_SERVER_CONGESTION_STATUS message is sent when the number of bytes in the write buffer reaches the *low_water* number.

The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.

Each MON_SERVER_CONGESTION_STATUS message contains five fields:

- an INT4 field containing the current size of the write buffer
- an INT4 field containing the threshold value passed (either the *high_water* or *low_water* number)
- a STR field containing the *server_name*
- a STR field containing the *connected_process_name*
- a BOOL field containing which threshold was passed (TRUE for the *high_water* threshold, FALSE if the *low_water* threshold was passed)

Once the number of bytes in the write buffer has reached the *high_water* number, RTserver does not send another MON_SERVER_CONGESTION_STATUS message until number of pending bytes in the buffer decreases to the *low_water* number.

Caution TIPCMonServerCongestionSetWatch sends a MON_SERVER_CONGESTION_SET_WATCH message to RTserver, but does not explicitly flush the message. See TIPC_SrvGetAutoFlushSize for more information on message buffering.

Write buffer congestion can only be watched, not polled.

See Also TIPCMonServerCongestionGetWatch, TIPCMonClientCongestionSetWatch, TIPCMonClientCongestionGetWatch

Examples This example uses `TipcMonServerCongestionSetWatch` to start watching buffer congestion in all connection buffers in all RTservers. The *high_water* threshold is 8 million bytes and the *low_water* threshold is 2 million bytes. The example creates a callback to process incoming `MON_SERVER_CONGESTION_STATUS` messages:

```
/* ===== */
/*..process_mon_server_congestion_status – process a
MON_SERVER_CONGESTION_STATUS message */

void process_mon_server_congestion_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_INT4 size;
    T_INT4 threshold;
    T_STR server_name;
    T_STR connected_process_name;
    T_BOOL high_water;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg,
                    T_IPC_FT_INT4, &size,
                    T_IPC_FT_INT4, &threshold,
                    T_IPC_FT_STR, &server_name,
                    T_IPC_FT_STR, &connected_process_name,
                    T_IPC_FT_BOOL, &high_water,
                    NULL)) {
        return; /* error */
    }

    TutOut("Got SERVER_CONGESTION status.\n");
    TutOut("Crossed %s water threshold.\n", high_water ? "high" :
"low");
    TutOut("RTserver name = %s\n", server_name);
    TutOut("connected process name = %s\n", connected_process_name);
    TutOut("buffer size = %d\n", size);
    TutOut("threshold = %d\n", threshold);
} /* process_mon_server_congestion_status */
```



```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonServerCongestionSetWatch ("/...", T_IPC_MON_ALL,
8000000,
2000000, TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_CONGESTION_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
process_mon_server_congestion_status,
NULL) == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonServerConnGetWatch

Name	TipcMonServerConnGetWatch — determine if this RTclient is watching RTserver connections
Synopsis	<pre>T_BOOL TipcMonServerConnGetWatch(<i>watch_status_return</i>) T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the RTserver connections watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonServerConnGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>watch_status_return</i> was null
Description	TipcMonServerConnGetWatch gets whether this RTclient is watching RTserver connections of the RTservers that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_SERVER_CONN_STATUS message each time a connection between RTservers is created or destroyed. Each MON_SERVER_CONN_STATUS message contains eight fields (all the array fields have the same number of elements): • a STR_ARRAY field containing the names of the first RTservers on the connections • a STR_ARRAY field containing the names of the second RTservers on the connections • a STR_ARRAY field containing the logical connection names of the connections • an INT4_ARRAY field containing the weights of the connections (always one) • a STR field containing the name of the first RTserver on the created connection (or "" if not applicable) • a STR field containing the name of the second RTserver on the created connection (or "" if not applicable) • a STR field containing the name of the first RTserver on the destroyed connection (or "" if not applicable) • a STR field containing the name of the second RTserver on the destroyed connection (or "" if not applicable)

To set whether this RTclient is watching RTserver connections, use `TipcMonServerConnSetWatch`. To poll once for RTserver connections, use `TipcMonServerConnPoll`.

For an example of how `MON_SERVER_CONN_STATUS` messages can be used, see the RTmon GDI Watch Server Connections window.

Caution If `TipcMonServerConnGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

See Also `TipcMonServerConnPoll`, `TipcMonServerConnSetWatch`

Examples This example uses `TipcMonServerConnGetWatch` to get the status of watching RTserver connections:

```
T_BOOL watch_status;

if (!TipcMonServerConnGetWatch(&watch_status)) {
    return; /* error */
}
TutOut("This process is %s RTserver connections.\n",
       watch_status ? "watching" : "not watching");
```

TipcMonServerConnPoll

Name	TipcMonServerConnPoll — poll once for RTserver connections
Synopsis	<code>T_BOOL TipcMonServerConnPoll()</code>
Arguments	None
Return Values	TRUE if the poll for RTserver connections was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonServerConnPoll fails, it returns FALSE and sets the global SmartSockets error number to: • any error number from TipcSrvMsgWrite
Description	TipcMonServerConnPoll polls for RTserver connections of the RTservers that match the value of the Monitor_Scope option by sending a MON_SERVER_CONN_POLL_CALL message to RTserver. RTserver responds by sending back a MON_SERVER_CONN_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. The MON_SERVER_CONN_POLL_RESULT message contains four fields (all the array fields have the same number of elements): • a STR_ARRAY field containing the names of the first RTservers on the connections • a STR_ARRAY field containing the names of the second RTservers on the connections • a STR_ARRAY field containing the logical connection names of the connections • an INT4_ARRAY field containing the weights of the connections (always one) To watch for created and destroyed RTserver connections, use TipcMonServerConnSetWatch.
Caution	TipcMonServerConnPoll sends a MON_SERVER_CONN_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.
See Also	TipcMonServerConnGetWatch, TipcMonServerConnSetWatch, TipcSrvMsgWrite

Examples This example uses TipcMonServerConnPoll to poll for RTserver connections, waits up to 10 seconds for the poll result, and then prints the poll result message:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *server1_names;
T_INT4 num_fields;
T_STR *server2_names;
T_STR *conn_names;
T_INT4 *weights;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcMonServerConnPoll()) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_CONN_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvMsgSearchType(10.0, mt);
if (msg == NULL) {
    return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}

/* all fields contains the same number of elements */
if (!TipcMsgNextStrArray(msg, &server1_names, &num_fields)
    || !TipcMsgNextStrArray(msg, &server2_names, &num_fields)
    || !TipcMsgNextStrArray(msg, &conn_names, &num_fields)
    || !TipcMsgNextInt4Array(msg, &weights, &num_fields)) {
    return; /* error */
}

TutOut("Got SERVER_CONN poll result.\n");
for (counter = 0; counter < num_fields; counter++) {
    TutOut("server1_names[%d] = %s\n", counter,
server1_names[counter]);
    TutOut("server2_names[%d] = %s\n", counter,
server2_names[counter]);
    TutOut("conn_names[%d] = %s\n", counter, conn_names[counter]);
    TutOut("weights[%d] = %d\n", counter, weights[counter]);
}
/* clean up */
if (!TipcMsgDestroy(msg)) {
    return; /* error */
}

```

TipcMonServerConnSetWatch

Name	TipcMonServerConnSetWatch — start or stop watching RTserver connections
Synopsis	<pre>T_BOOL TipcMonServerConnSetWatch(<i>watch_status</i>) T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the RTserver connections watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonServerConnSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonServerConnSetWatch sets whether this RTclient is watching RTserver connections of the RTservers that match the value of the Monitor_Scope option. TipcMonServerConnSetWatch saves the watch status and sends a MON_SERVER_CONN_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_SERVER_CONN_STATUS message each time a connection between RTservers is created or destroyed. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. Each MON_SERVER_CONN_STATUS message contains eight fields (all the array fields have the same number of elements): • a STR_ARRAY field containing the names of the first RTservers on the connections • a STR_ARRAY field containing the names of the second RTservers on the connections • a STR_ARRAY field containing the logical connection names of the connections • an INT4_ARRAY field containing the weights of the connections (always one) • a STR field containing the name of the first RTserver on the created connection (or "" if not applicable) • a STR field containing the name of the second RTserver on the created connection (or "" if not applicable)

- a STR field containing the name of the first RTserver on the destroyed connection (or "" if not applicable)
- a STR field containing the name of the second RTserver on the destroyed connection (or "" if not applicable)

In addition to sending a MON_SERVER_CONN_STATUS message each time an RTserver connection is created or destroyed, RTserver also sends an initial MON_SERVER_CONN_STATUS message when the watch status is set to TRUE, so that the RTclient does not have to wait for the next RTserver connection change to receive the first MON_SERVER_CONN_STATUS message.

To determine if this RTclient is watching RTserver connections, use TipcMonServerConnGetWatch. To poll once for RTserver connections, use TipcMonServerConnPoll.

For an example of how MON_SERVER_CONN_STATUS messages can be used, see the RTmon GDI Watch Server Connections window.

Caution TipcMonServerConnSetWatch sends a MON_SERVER_CONN_SET_WATCH message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

See Also TipcMonServerConnGetWatch, TipcMonServerConnPoll

Examples This example uses TipcMonServerConnSetWatch to start watching RTserver connections and creates a callback to process the incoming MON_SERVER_CONN_STATUS messages:

```
/* ===== */
/*..process_mon_server_conn_status -- process a MON_SERVER_CONN_STATUS message */
void T_ENTRY process_mon_server_conn_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR *server1_names;
    T_INT4 num_fields;
    T_STR *server2_names;
    T_STR *conn_names;
    T_INT4 *weights;
    T_STR created_server1;
    T_STR created_server2;
    T_STR destroyed_server1;
    T_STR destroyed_server2;
    T_INT4 counter;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }
}
```

```

    /* get the fields from the message */
    /* the array fields contain the same number of elements */
    if (!TipcMsgNextStrArray(data->msg, &server1_names,
                             &num_fields)
        || !TipcMsgNextStrArray(data->msg, &server2_names,
                                 &num_fields)
        || !TipcMsgNextStrArray(data->msg, &conn_names,
                                 &num_fields)
        || !TipcMsgNextInt4Array(data->msg, &weights,
                                 &num_fields)
        || !TipcMsgNextStr(data->msg, &created_server1)
        || !TipcMsgNextStr(data->msg, &created_server2)
        || !TipcMsgNextStr(data->msg, &destroyed_server1)
        || !TipcMsgNextStr(data->msg, &destroyed_server2)) {
        return; /* error */
    }

    TutOut("Got SERVER_CONN status.\n");
    for (counter = 0; counter < num_fields; counter++) {
        TutOut("server1_names[%d] = %s\n", counter,
              server1_names[counter]);
        TutOut("server2_names[%d] = %s\n", counter,
              server2_names[counter]);
        TutOut("conn_names[%d] = %s\n", counter,
              conn_names[counter]);
        TutOut("weights[%d] = %d\n", counter,
              weights[counter]);
    }
    TutOut("created RTserver 1 = %s\n", created_server1);
    TutOut("created RTserver 2 = %s\n", created_server2);
    TutOut("destroyed RTserver 1 = %s\n", destroyed_server1);
    TutOut("destroyed RTserver 2 = %s\n", destroyed_server2);
} /* process_mon_server_conn_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcMonServerConnSetWatch(TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_CONN_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_mon_server_conn_status,
NULL)
    == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```


TipcMonServerCpuPoll

Name	TipcMonServerCpuPoll — poll for the percentage of CPU time used by an RTserver
Synopsis	<pre>T_BOOL TipcMonServerCpuPoll(server_name) T_STR server_name;</pre>
Arguments	<i>server_name</i> — name of RTserver to get CPU usage for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)
Return Values	TRUE if the poll for the number of subjects was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonServerCpuPoll fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>server_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonServerCpuPoll polls for the percentage of CPU time used by an RTserver by sending a MON_SERVER_CPU_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTservers respond by sending back a MON_SERVER_CPU_POLL_RESULT message. Each MON_SERVER_CPU_POLL_RESULT contains two fields: • a STR field containing the RTserver name • a REAL4 field containing the percentage of CPU time used (between 0.0 and 100.0) There are multiple responses only if <i>server_name</i> is a wildcard or T_IPC_MON_ALL. If the RTserver cannot query the operating system, the value reported in the CPU percentage used field is a negative number.
Caution	There is no response message if the RTserver named <i>server_name</i> does not exist; the poll is silently dropped.

TipcMonServerCpuPoll sends a MON_SERVER_CPU_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

RTserver CPU usage information can only be polled, not watched.

See Also TipcMonClientCpuPoll

Examples This example uses TipcMonServerCpuPoll to poll for information in all RTservers, creates a callback to process the incoming MON_SERVER_CPU_POLL_RESULT messages, and waits up to 10 seconds for the poll results.:

```
/* ===== */
/*..process_mon_server_cpu_poll_result - process a
MON_SERVER_CPU_POLL_RESULT message */

void process_mon_server_cpu_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR server_name;
    T_REAL4 cpu_utilization;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg,
                     T_IPC_FT_STR, &server_name,
                     T_IPC_FT_REAL4, &cpu_utilization,
                     NULL)) {
        return; /* error */
    }

    TutOut("Got SERVER_CPU poll result.\n");
    TutOut("RTserver name = %s\n", server_name);
    TutOut("CPU utilization percentage = %f\n", cpu_utilization);
} /* process_mon_server_cpu_poll_result */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonServerCpuPoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_CPU_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_mon_server_cpu_poll_result,
                          NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming messages) that arrive within the
next 10 seconds */

if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonServerGeneralPoll

Name	TipcMonServerGeneralPoll — poll once for RTserver general information
Synopsis	<pre>T_BOOL TipcMonServerGeneralPoll(server_name) T_STR server_name;</pre>
Arguments	<i>server_name</i> — name of RTserver to poll for general information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)
Return Values	TRUE if the poll for RTserver general information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonServerGeneralPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: - T_ERR_NULL_PTR — <i>server_name</i> was null - any error number from TipcSrvMsgWrite
Description	TipcMonServerGeneralPoll polls for RTserver general information by sending a MON_SERVER_GENERAL_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTserver responds by sending back a MON_SERVER_GENERAL_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_SERVER_GENERAL_POLL_RESULT message contains seventeen fields: - a STR field containing the name of the RTserver - a STR field containing the process monitoring identifier (defaults to a major type of license, such as RTserver) - a STR field containing the node name the RTserver is running on - a STR field containing the user name the RTserver is running under - an INT4 field containing the PID of the RTserver - a STR field containing the name of the SmartSockets architecture (such as sun4_solaris)

- an INT4 field containing the current operating system sbrk value (the edge of the RTserver's virtual memory address space)
- an INT4 field containing the amount of change in the current operating system sbrk value (the edge of the process's virtual memory address space) since the RTserver started (actually, since the process's first call to any SmartSockets function)
- an INT2 field containing the integer number format of the RTserver (really of type T_INT_FORMAT, but stored in an INT2 field to save space)
- an INT2 field containing the real number format of the RTserver (really of type T_REAL_FORMAT, but stored in an INT2 field to save space)
- a STR field containing the name of the startup command file that the RTserver used at initialization
- an INT4 field containing whether the RTserver was started with the `-no_daemon` command-line argument (really of type BOOL)
- an INT4 field containing whether the RTserver was started with the `-cmd_mode` command-line argument (really of type BOOL)
- a STR_ARRAY field containing the names of the RTclients in the same project as the polling RTclient which are directly connected to the RTserver
- a STR_ARRAY field containing the names of the RTservers which are directly connected to the RTserver
- a STR_ARRAY field containing the names of the subjects in the same project as the polling RTclient to which the RTserver is subscribing as the result of an RTserver subscribe command
- a STR_ARRAY field containing the names of the subjects in the same project as the polling RTclient to which the RTserver is subscribing as the result of direct RTclients subscriptions

There are multiple responses only if `server_name` is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTclient general information, use `TipcMonClientGeneralPoll`.

For an example of how `MON_SERVER_GENERAL_POLL_RESULT` messages can be used, see the RTmon GDI Server Information window.

Caution TipcMonServerGeneralPoll sends a MON_SERVER_GENERAL_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no response message if the RTserver named *server_name* does not exist; the poll is silently dropped.

RTserver general information can only be polled, not watched.

See Also TipcMonClientGeneralPoll, TipcSrvMsgWrite

Examples This example uses TipcMonServerGeneralPoll to poll for general information in all RTservers, creates a callback to process the incoming MON_SERVER_GENERAL_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_general_poll_result -- process a
MON_SERVER_GENERAL_POLL_RESULT message */
void T_ENTRY process_mon_server_general_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR server_name;
    T_STR ident;
    T_STR node_name;
    T_STR user_name;
    T_INT4 pid;
    T_STR arch;
    T_INT4 current_sbrk;
    T_INT4 delta_sbrk;
    T_INT2 int_format;
    T_INT2 real_format;
    T_STR cmd_file_name;
    T_INT4 no_daemon_flag;
    T_INT4 cmd_mode_flag;
    T_STR *direct_client_names;
    T_INT4 num_direct_client_names;
    T_STR *direct_server_names;
    T_INT4 num_direct_server_names;
    T_STR *server_subscribes;
    T_INT4 num_server_subscribes;
    T_STR *client_subscribes;
    T_INT4 num_client_subscribes;
    T_INT4 counter;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }
}
```

```

/* get the fields from the message */
if (!TipcMsgRead(data->msg,
    T_IPC_FT_STR, &server_name,
    T_IPC_FT_STR, &ident,
    T_IPC_FT_STR, &node_name,
    T_IPC_FT_STR, &user_name,
    T_IPC_FT_INT4, &pid,
    T_IPC_FT_STR, &arch,
    T_IPC_FT_INT4, &current_sbrk,
    T_IPC_FT_INT4, &delta_sbrk,
    T_IPC_FT_INT2, &int_format,
    T_IPC_FT_INT2, &real_format,
    T_IPC_FT_STR, &cmd_file_name,
    T_IPC_FT_INT4, &no_daemon_flag,
    T_IPC_FT_INT4, &cmd_mode_flag,
    T_IPC_FT_STR_ARRAY, &direct_client_names,
    &num_direct_client_names,
    T_IPC_FT_STR_ARRAY, &direct_server_names,
    &num_direct_server_names,
    T_IPC_FT_STR_ARRAY, &server_subscribes,
    &num_server_subscribes,
    T_IPC_FT_STR_ARRAY, &client_subscribes,
    &num_client_subscribes,
    NULL)) {
    return; /* error */
}

TutOut("Got SERVER_GENERAL poll result.\n");
TutOut("RTserver name = %s\n", server_name);
TutOut("ident = %s\n", ident);
TutOut("node name = %s\n", node_name);
TutOut("user name = %s\n", user_name);
TutOut("pid = %d\n", pid);
TutOut("architecture = %s\n", arch);
TutOut("current sbrk value = %d\n", current_sbrk);
TutOut("delta sbrk value = %d\n", delta_sbrk);
TutOut("int format = %d\n", int_format);
TutOut("real format = %d\n", real_format);
TutOut("cmd file name = %s\n", cmd_file_name);
TutOut("-no_daemon flag = %s\n", no_daemon_flag ? "TRUE" :
                                              "FALSE");
TutOut("-cmd_mode flag = %s\n", cmd_mode_flag ? "TRUE" :
                                              "FALSE");

TutOut(" direct client names:");
for (counter = 0; counter < num_direct_client_names; counter++) {
    TutOut(" %s", direct_client_names[counter]);
}
TutOut("\n");

TutOut(" direct server names:");
for (counter = 0; counter < num_direct_server_names; counter++) {
    TutOut(" %s", direct_server_names[counter]);
}
TutOut("\n");

```

```

    TutOut("  server subscribes:");
    for (counter = 0; counter < num_server_subscribes; counter++) {
        TutOut(" %s", server_subscribes[counter]);
    }
    TutOut("\n");

    TutOut("  client subscribes:");
    for (counter = 0; counter < num_client_subscribes; counter++) {
        TutOut(" %s", client_subscribes[counter]);
    }
    TutOut("\n");

} /* process_mon_server_general_poll_result */

/*===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonServerGeneralPoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_GENERAL_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}
if (TipcSrvProcessCbCreate(mt,
    process_mon_server_general_poll_result,
    NULL) == NULL) {
    return; /* error */
}
/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```


TipcMonServerMaxClientLicensesGetWatch

Name	TipcMonServerMaxClientLicensesGetWatch — determine if the RTserver is watching for the server cloud to exceed the licensed number of RTclient connections
Synopsis	<pre>T_BOOL TipcMonServerMaxClientLicensesGetWatch(server_name, watch_status_return) T_STR server_name; T_BOOL *watch_status_return;</pre>
Arguments	<i>server_name</i> — name of the RTserver to set watch status (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope) <i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonServerMaxClientLicensesGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>server_name</i> or <i>watch_status_return</i> was null
Description	TipcMonServerMaxClientLicensesGetWatch determines whether the <i>server_name</i> RTserver is watching for the server cloud to exceed the maximum licensed number of RTclient connections. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The <i>watch_status_return</i> argument is set to TRUE if the watch is currently active, FALSE if not.
Caution	TipcMonServerMaxClientLicensesGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i>. RTserver licenses can only be watched, not polled.
See Also	TipcMonServerMaxClientLicensesSetWatch

Examples This example uses `TipcMonServerMaxClientLicensesGetWatch` to determine whether the RTserver is watching for the RTserver cloud to exceed the maximum licensed number of RTclient connections:

```
T_BOOL watch_status;

if (!TipcMonServerMaxClientLicensesGetWatch("/...",
&watch_status)) {
    return; /* error */
}
TutOut("This RTclient is %s client rejections in ",
      watch_status ? "watching" : "not watching");
TutOut("all servers.\n");
```

TipcMonServerMaxClientLicensesSetWatch

Name	TipcMonServerMaxClientLicensesSetWatch — start or stop watching for the RTserver cloud to exceed the licensed number of RTclient connections
Synopsis	<pre>T_BOOL TipcMonServerMaxClientLicensesSetWatch(server_name, watch_status) T_STR server_name; T_BOOL watch_status;</pre>
Arguments	<i>server_name</i> — name of RTserver to set watch status (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope) <i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonServerMaxClientLicensesSetWatch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>server_name</i> was null • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonServerMaxClientLicensesSetWatch sets whether the RTserver is watching for the RTserver cloud to exceed the maximum licensed number of RTclient connections. TipcMonServerMaxClientLicensesSetWatch saves the watch status and sends a MON_SERVER_MAX_CLIENT_LICENSES_SET_WATCH message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_SERVER_MAX_CLIENT_LICENSES_STATUS message each time the RTserver exceeds the allowed number of client connections and refuses a new RTclient connection. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.

Each `MON_SERVER_MAX_CLIENT_LICENSES_STATUS` contains three fields:

- an `INT4` field containing the number of `RTclient` connections licensed
- a `STR` field containing the name of the `RTclient` that was refused a connection
- a `STR` field containing the name of the `RTserver` that refused the connection

Caution `TipcMonServerMaxClientLicensesSetWatch` sends a `MON_SERVER_MAX_CLIENT_LICENSES_SET_WATCH` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

`RTserver` licenses can only be watched, not polled.

See Also `TipcMonServerMaxClientLicensesGetWatch`

Examples This example uses `TipcMonServerMaxClientLicensesSetWatch` to start watching all `RTservers` for client connections that were rejected because the maximum number of client licenses is exceeded, and creates a callback to process the incoming `MON_SERVER_MAX_CLIENT_LICENSES_STATUS` messages:

```
/* ===== */
/*..process_mon_server_max_client_licenses_status - process a
MON_SERVER_MAX_CLIENT_LICENSESESE_STATUS message */

void process_mon_server_max_client_licenses_status(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_INT4 max_licenses;
    T_STR client_name;
    T_STR server_name;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg,
                     T_IPC_FT_INT4, &max_licenses,
                     T_IPC_FT_STR, &client_name,
                     T_IPC_FT_STR, &server_name,
                     NULL)) {
        return; /* error */
    }

    TutOut("Got SERVER_MAX_CLIENT_LICENSES status.\n");
    TutOut("rejected RTclient name = %s\n", client_name);
    TutOut("rejecting RTserver name = %s\n", server_name);
} /* process_mon_server_max_client_licenses_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonServerMaxClientLicensesSetWatch ("/...", TRUE))
    return; /* error */
}

mt =
TipcMtLookupByNum(T_MT_MON_SERVER_MAX_CLIENT_LICENSES_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
process_mon_server_max_client_licenses_status,
NULL) == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonServerMsgTrafficExPoll

Name TipcMonServerMsgTrafficExPoll — poll once for RTserver message traffic information



This new function supersedes TipcMonServerMsgTrafficPoll for clients that use release 6.7 or later.

Synopsis `T_BOOL TipcMonServerMsgTrafficExPoll(server_name, connected_process_name)
T_STR server_name;
T_STR connected_process_name;`

Arguments *server_name* — name of RTserver to poll for message traffic information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)

connected_process_name — name of connected process whose message traffic in RTserver to poll (use T_IPC_MON_ALL to indicate a poll of all processes that the polled RTserver is connected to)

Return Values TRUE if the poll for RTserver message traffic information was successfully initiated, FALSE otherwise.

Diagnostics If TipcMonServerMsgTrafficExPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *server_name* or *connected_process_name* was null
- any error number from TipcSrvMsgWrite

Description TipcMonServerMsgTrafficExPoll polls for RTserver message traffic information by sending a MON_SERVER_MSG_TRAFFIC_EX_POLL_CALL message to the *server_name* RTserver. The *server_name* argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The message traffic information of *connected_process_name* (which can be either an RTserver or another RTclient in the same project) is polled. The polled RTserver responds by sending back a MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT message. The response should come back quickly. If *connected_process_name* is T_IPC_MON_ALL then the message traffic in the polled RTserver of all connected processes are polled. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response.

Each `MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT` message contains one fixed field followed by five array fields. The fixed field is a `STR` field containing the name of the RTserver.

The array fields are:

- a `STR_ARRAY` field containing the name of the process that the RTserver is connected to
- an `INT8_ARRAY` field containing the number of messages the polled RTserver has sent on its connection to each connected process
- an `INT8_ARRAY` field containing the number of messages the polled RTserver has received on its connection to each connected process
- an `INT8_ARRAY` field containing the number of bytes the polled RTserver has sent on its connection to each connected process
- an `INT8_ARRAY` field containing the number of bytes the polled RTserver has received on its connection to each connected process

There are multiple responses only if `server_name` is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTclient message traffic information, use `TipcMonClientMsgTrafficPoll`.

For an example of how `MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT` messages can be used, see the RTmon GDI Watch Server Connections window.

Caution `TipcMonServerMsgTrafficPoll` sends a `MON_SERVER_MSG_TRAFFIC_EX_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTserver named `server_name` does not exist; the poll is silently dropped.

RTserver message traffic information can only be polled, not watched.

See Also `TipcMonClientMsgTrafficPoll`, `TipcSrvMsgWrite`

Examples This example uses TipcMonServerMsgTrafficExPoll to poll for all message traffic information in all RTservers, creates a callback to process the incoming MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_msg_traffic_ex_poll_result -- process a
MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT message */

void process_mon_server_msg_traffic_ex_poll_result
(
    T_IPC_CONN conn,
    T_IPC_CONN_PROCESS_CB_DATA data,
    T_CB_ARG arg
)
{
    T_MSG    msg;
    T_STR    server_name;
    T_STR    *process_name;
    T_INT4    num_processes;
    T_INT8    *total_msg_send;
    T_INT4    num_msg_send;
    T_INT8    *total_msg_rcv;
    T_INT4    num_msg_rcv;
    T_INT8    *total_byte_send;
    T_INT4    num_byte_send;
    T_INT8    *total_byte_rcv;
    T_INT4    num_byte_rcv;
    T_INT4    counter;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    msg = data->msg;
    if (!TipcMsgNextStr(msg, &server_name)
        || !TipcMsgNextStrArray(msg, &process_name, &num_processes)
        || !TipcMsgNextInt8Array(msg, &total_msg_send, &num_msg_send)
        || !TipcMsgNextInt8Array(msg, &total_msg_rcv, &num_msg_rcv)
        || !TipcMsgNextInt8Array(msg, &total_byte_send, &num_byte_send)
        || !TipcMsgNextInt8Array(msg, &total_byte_rcv, &num_byte_rcv))
    {
        return; /* error */
    }
    TutOut("Got SERVER_MSG_TRAFFIC_EX poll result.\n");
    TutOut("RTserver name = %s\n", server_name);

    /* loop through all groups */
    for (counter=0; counter<num_processes; counter++) {
        TutOut("connected process name = %s\n", process_name[counter]);
    }
}
```



```

/*
 * IMPORTANT: The format string for T_INT8 differs by platform.
 * The T_INT8_SPEC utility macro supplies the correct format
 * string for each platform.
 */
    TutOut("total msg send = " T_INT8_SPEC "\n",
           total_msg_send[counter]);
    TutOut("total msg recv = " T_INT8_SPEC "\n",
           total_msg_recv[counter]);
    TutOut("total byte send = " T_INT8_SPEC "\n",
           total_byte_send[counter]);
    TutOut("total byte recv = " T_INT8_SPEC "\n",
           total_byte_recv[counter]);
}

} /* process_mon_server_msg_traffic_ex_poll_result */

/*===== */
/*...code from calling function is below */

T_IPC_MT mt;

mt =
TipcMtLookupByNum(T_MT_MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}
if (TipcSrvProcessCbCreate(mt,

process_mon_server_msg_traffic_ex_poll_result,
                           NULL) == NULL) {

    return; /* error */
}

/* send the poll request out to RTserver */
if (!TipcMonServerMsgTrafficExPoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonServerMsgTrafficPoll

Name TipcMonServerMsgTrafficPoll — poll once for RTserver message traffic information

 **Obsolete**

The previous function, `TipcMonServerMsgTrafficExPoll`, supersedes this function for all new development. Use this older function only when operating with clients prior to release 6.7. This poll can truncate the statistics data that it returns.

Synopsis `T_BOOL TipcMonServerMsgTrafficPoll(server_name, connected_process_name)`
`T_STR server_name;`
`T_STR connected_process_name;`

Arguments *server_name* — name of RTserver to poll for message traffic information (wildcard names allowed, or use `T_IPC_MON_ALL` to indicate a poll of all RTservers matching the value of `Monitor_Scope`)

connected_process_name — name of connected process whose message traffic in RTserver to poll (use `T_IPC_MON_ALL` to indicate a poll of all processes that the polled RTserver is connected to)

Return Values TRUE if the poll for RTserver message traffic information was successfully initiated, FALSE otherwise.

Diagnostics If `TipcMonServerMsgTrafficPoll` fails, it returns FALSE and sets the global SmartSockets error number to one of:

- `T_ERR_NULL_PTR` — *server_name* or *connected_process_name* was null
- any error number from `TipcSrvMsgWrite`

Description `TipcMonServerMsgTrafficPoll` polls for RTserver message traffic information by sending a `MON_SERVER_MSG_TRAFFIC_POLL_CALL` message to the *server_name* RTserver. The *server_name* argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or `T_IPC_MON_ALL` to poll all RTservers that match the value of the `Monitor_Scope` option. The message traffic information of *connected_process_name* (which can be either an RTserver or another RTclient in the same project) is polled. The polled RTserver responds by sending back a `MON_SERVER_MSG_TRAFFIC_POLL_RESULT` message. The response should come back quickly. If *connected_process_name* is `T_IPC_MON_ALL` then the message traffic in the polled RTserver of all connected processes are polled. The polling RTclient can use functions such as `TipcSrvMsgSearchType` or `TipcSrvMainLoop` to get the response.

Each `MON_SERVER_MSG_TRAFFIC_POLL_RESULT` message contains one fixed field followed by one or more groups of five fields. The fixed field is a `STR` field containing the name of the RTserver.

The group of five fields is:

- a `STR` field containing the name of the process RTserver is connected to
- an `INT4` field containing the number of messages the polled RTserver has sent on its connection to the connected process
- an `INT4` field containing the number of messages the polled RTserver has received on its connection to the connected process
- an `INT4` field containing the number of bytes the polled RTserver has sent on its connection to the connected process
- an `INT4` field containing the number of bytes the polled RTserver has received on its connection to the connected process

There are multiple responses only if `server_name` is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTclient message traffic information, use `TipcMonClientMsgTrafficPoll`.

For an example of how `MON_SERVER_MSG_TRAFFIC_POLL_RESULT` messages can be used, see the RTmon GDI Watch Server Connections window.

Caution `TipcMonServerMsgTrafficPoll` sends a `MON_SERVER_MSG_TRAFFIC_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTserver named `server_name` does not exist; the poll is silently dropped.

RTserver message traffic information can only be polled, not watched.

See Also `TipcMonClientMsgTrafficPoll`, `TipcSrvMsgWrite`

Examples This example uses TipcMonServerMsgTrafficPoll to poll for all message traffic information in all RTservers, creates a callback to process the incoming MON_SERVER_MSG_TRAFFIC_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_msg_traffic_poll_result -- process a
MON_SERVER_MSG_TRAFFIC_POLL_RESULT message */
void T_ENTRY process_mon_server_msg_traffic_poll_result(conn,
data,
arg)

T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR server_name;
    T_STR connected_process_name;
    T_INT4 total_msg_send;
    T_INT4 total_msg_recv;
    T_INT4 total_byte_send;
    T_INT4 total_byte_recv;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get first fixed field */
    if (!TipcMsgNextStr(data->msg, &server_name)) {
        return; /* error */
    }
    TutOut("Got SERVER_MSG_TRAFFIC poll result.\n");
    TutOut("RTserver name = %s\n", server_name);

    /* loop through all groups */
    while (TipcMsgRead(data->msg,
        T_IPC_FT_STR, &connected_process_name,
        T_IPC_FT_INT4, &total_msg_send,
        T_IPC_FT_INT4, &total_msg_recv,
        T_IPC_FT_INT4, &total_byte_send,
        T_IPC_FT_INT4, &total_byte_recv,
        NULL)) {
        TutOut("connected process name = %s\n",
connected_process_name);
        TutOut("total msg send = %d\n", total_msg_send);
        TutOut("total msg recv = %d\n", total_msg_recv);
        TutOut("total byte send = %d\n", total_byte_send);
        TutOut("total byte recv = %d\n", total_byte_recv);
    }

    /* make sure we reached the end of the message */
    if (TutErrNumGet() != T_ERR_MSG_EOM) {
        return; /* error */
    }
} /* process_mon_server_msg_traffic_poll_result */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonServerMsgTrafficPoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_MSG_TRAFFIC_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
                           process_mon_server_msg_traffic_poll_result,
                           NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonServerNamesGetWatch

Name	TipcMonServerNamesGetWatch — determine if this RTclient is watching RTserver names
Synopsis	<pre>T_BOOL TipcMonServerNamesGetWatch(<i>watch_status_return</i>) T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the RTserver names watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonServerNamesGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>watch_status_return</i> was null
Description	TipcMonServerNamesGetWatch gets whether this RTclient is watching the RTserver names that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_SERVER_NAMES_STATUS message each time an RTserver is created or destroyed. An RTserver is considered created when it starts up (and usually connects to other RTservers). An RTserver is considered destroyed when it terminates or disconnects from other RTservers. Each MON_SERVER_NAMES_STATUS message contains four fields: • a STR_ARRAY field containing the current RTserver names • a STR_ARRAY field containing RTserver info strings of the form "<i>ident: user@node</i>" (for example, "Primary_RTclient: ssuser@workstation1") • a STR field containing the name of the created RTserver (or "" if not applicable) • a STR field containing the name of the destroyed RTserver (or "" if not applicable) To set whether this RTclient is watching RTserver names, use TipcMonServerNamesSetWatch. To poll once for RTserver names, use TipcMonServerNamesPoll.
Caution	If TipcMonServerNamesGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i> .
See Also	TipcMonServerNamesPoll, TipcMonServerNamesSetWatch

Examples This example uses TipcMonServerNamesGetWatch to get the status of watching RTserver names:

```
T_BOOL watch_status;

if (!TipcMonServerNamesGetWatch(&watch_status)) {
    return; /* error */
}
TutOut("This process is %s RTserver names.\n",
       watch_status ? "watching" : "not watching");
```

TipcMonServerNamesPoll

Name	TipcMonServerNamesPoll — poll once for RTserver names
Synopsis	<code>T_BOOL TipcMonServerNamesPoll()</code>
Arguments	None
Return Values	TRUE if the poll for RTserver names was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonServerNamesPoll fails, it returns FALSE and sets the global SmartSockets error number to: • any error number from TipcSrvMsgWrite
Description	TipcMonServerNamesPoll polls for the RTserver names that match the value of the Monitor_Scope option by sending a MON_SERVER_NAMES_POLL_CALL message to RTserver. RTserver responds by sending back a MON_SERVER_NAMES_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. The MON_SERVER_NAMES_POLL_RESULT message contains two fields: • a STR_ARRAY field containing the current RTserver names • a STR_ARRAY field containing RTserver info strings of the form "<i>ident: user@node</i>" (for example, "RTserver: ssuser@workstation1") To watch for created and destroyed RTserver names, use TipcMonServerNamesSetWatch.
Caution	TipcMonServerNamesPoll sends a MON_SERVER_NAMES_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.
See Also	TipcMonServerNamesGetWatch, TipcMonServerNamesSetWatch, TipcSrvMsgWrite

Examples This example uses TipcMonServerNamesPoll to poll for RTserver names, waits up to 10 seconds for the poll result, and then prints the poll result message:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *server_names;
T_INT4 num_server_names;
T_STR *info_strs;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcMonServerNamesPoll()) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_NAMES_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvMsgSearchType(10.0, mt);
if (msg == NULL) {
    return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}

/* both server_names and info_strs contains the same number of elements */
if (!TipcMsgNextStrArray(msg, &server_names, &num_server_names)
    || !TipcMsgNextStrArray(msg, &info_strs, &num_server_names)) {
    return; /* error */
}

TutOut("Got SERVER_NAMES poll result.\n");
for (counter = 0; counter < num_server_names; counter++) {
    TutOut("server_names[%d] = %s\n", counter,
server_names[counter]);
    TutOut("info_strs[%d] = %s\n", counter, info_strs[counter]);
}

/* clean up */
if (!TipcMsgDestroy(msg)) {
    return; /* error */
}

```

TipcMonServerNamesSetWatch

Name	TipcMonServerNamesSetWatch — start or stop watching RTserver names
Synopsis	<pre>T_BOOL TipcMonServerNamesSetWatch(<i>watch_status</i>) T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the RTserver names watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonServerNamesSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonServerNamesSetWatch sets whether this RTclient is watching the RTserver names that match the value of the Monitor_Scope option. TipcMonServerNamesSetWatch saves the watch status and sends a MON_SERVER_NAMES_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_SERVER_NAMES_STATUS message each time an RTserver is created or destroyed. An RTserver is considered created when it starts up (and usually connects to other RTservers). An RTserver is considered destroyed when it terminates or disconnects from other RTservers. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. Each MON_SERVER_NAMES_STATUS message contains four fields: • a STR_ARRAY field containing the current RTserver names • a STR_ARRAY field containing RTserver info strings of the form "<i>ident: user@node</i>" (for example, "RTserver: ssuser@workstation1") • a STR field containing the name of the created RTserver (or "" if not applicable) • a STR field containing the name of the destroyed RTserver (or "" if not applicable) In addition to sending a MON_SERVER_NAMES_STATUS message each time an RTserver is created or destroyed, RTserver also sends an initial MON_SERVER_NAMES_STATUS message when the watch status is set to TRUE, so that the RTclient does not have to wait for the next RTserver name change to receive the first MON_SERVER_NAMES_STATUS message.

To determine if this RTclient is watching RTserver names, use `TipcMonServerNamesGetWatch`. To poll once for RTserver names, use `TipcMonServerNamesPoll`.

Caution `TipcMonServerNamesSetWatch` sends a `MON_SERVER_NAMES_SET_WATCH` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

See Also `TipcMonServerNamesGetWatch`, `TipcMonServerNamesPoll`

Examples This example uses `TipcMonServerNamesSetWatch` to start watching RTserver names and creates a callback to process the incoming `MON_SERVER_NAMES_STATUS` messages:

```
/* ===== */
/*..process_mon_server_names_status -- process a MON_SERVER_NAMES_STATUS message */
void T_ENTRY process_mon_server_names_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR *server_names;
    T_INT4 num_server_names;
    T_STR *info_strs;
    T_STR created_server;
    T_STR destroyed_server;
    T_INT4 counter;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */

    /* both server_names and info_strs contains the same number of
    elements */

    if (!TipcMsgNextStrArray(data->msg, &server_names,
                             &num_server_names)
        || !TipcMsgNextStrArray(data->msg, &info_strs,
                                 &num_server_names)
        || !TipcMsgNextStr(data->msg, &created_server)
        || !TipcMsgNextStr(data->msg, &destroyed_server)) {
        return; /* error */
    }
}
```

```

    TutOut("Got SERVER_NAMES status.\n");
    for (counter = 0; counter < num_server_names; counter++) {
        TutOut("server_names[%d] = %s\n", counter,
server_names[counter]);
        TutOut("info_strs[%d] = %s\n", counter, info_strs[counter]);
    }
    TutOut("created RTserver = %s\n", created_server);
    TutOut("destroyed RTserver = %s\n", destroyed_server);
} /* process_mon_server_names_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcMonServerNamesSetWatch(TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_NAMES_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_mon_server_names_status,
NULL)
    == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonServerOptionPoll

Name	TipcMonServerOptionPoll — poll once for RTserver option information
Synopsis	<pre>T_BOOL TipcMonServerOptionPoll(server_name, option_name) T_STR server_name; T_STR option_name;</pre>
Arguments	<i>server_name</i> — name of RTserver to poll for option information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope) <i>option_name</i> — name of option to poll for information (use T_IPC_MON_ALL to indicate a poll of all options)
Return Values	TRUE if the poll for RTserver option information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonServerOptionPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>server_name</i> or <i>option_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonServerOptionPoll polls for RTserver option information by sending a MON_SERVER_OPTION_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTserver responds by sending back a MON_SERVER_OPTION_POLL_RESULT message. The response should come back quickly. If <i>option_name</i> is T_IPC_MON_ALL then all options in the polled RTserver are polled. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response.

Each `MON_SERVER_OPTION_POLL_RESULT` message contains one fixed field followed by one or more groups of five fields. The fixed field is a STR field containing the name of the RTserver. The group of five fields is:

- a STR field containing the name of the option
- an INT2 field containing the option type (really `T_OPT_TYPE`)
- a STR field containing the value of the option, converted to a string like the `setopt` command displays
- an INT4 field containing whether the option is required to have a value (that is, cannot be `UNKNOWN`)
- a `STR_ARRAY` field containing the legal values for the option; this array is empty if not applicable

There are multiple responses only if *server_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTclient option information, use `TipcMonClientOptionPoll`.

For an example of how `MON_SERVER_OPTION_POLL_RESULT` messages can be used, see the RTmon GDI Server Information window.

Caution `TipcMonServerOptionPoll` sends a `MON_SERVER_OPTION_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTserver named *server_name* or the option named *option_name* does not exist; the poll is silently dropped.

RTserver option information can only be polled, not watched.

See Also `TipcMonClientOptionPoll`, `TipcSrvMsgWrite`

Examples This example uses TipcMonServerOptionPoll to poll for all option information in all RTservers, creates a callback to process the incoming MON_SERVER_OPTION_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_option_poll_result -- process a
MON_SERVER_OPTION_POLL_RESULT message */
void T_ENTRY process_mon_server_option_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR server_name;
    T_STR option_name;
    T_INT2 type; /* really (T_OPT_TYPE) */
    T_STR value_str;
    T_INT4 required; /* really (T_BOOL) */
    T_STR *legal_vals;
    T_INT4 num_legal_vals; /* is zero if legal_vals not applicable */
    T_INT4 counter; /* loop counter */

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get first fixed field */
    if (!TipcMsgNextStr(data->msg, &server_name)) {
        return; /* error */
    }
    TutOut("Got SERVER_OPTION poll result.\n");
    TutOut("RTserver name = %s\n", server_name);
    /* loop through all groups */
    while (TipcMsgRead(data->msg,
        T_IPC_FT_STR, &option_name,
        T_IPC_FT_INT2, &type,
        T_IPC_FT_STR, &value_str,
        T_IPC_FT_INT4, &required,
        T_IPC_FT_STR_ARRAY, &legal_vals,
        &num_legal_vals,
        NULL)) {
        TutOut("option name = %s\n", option_name);
        TutOut("type = %d\n", type);
        TutOut("value str = %s\n", value_str);
        TutOut("required = %s\n", required ? "TRUE" : "FALSE");
        if (num_legal_vals > 0) {
            TutOut("legal vals:\n");
            for (counter = 0; counter < num_legal_vals; counter++) {
                TutOut(" %s\n", legal_vals[counter]);
            }
        }
    }
}
```

```

    /* make sure we reached the end of the message */
    if (TutErrNumGet() != T_ERR_MSG_EOM) {
        return; /* error */
    }
} /* process_mon_server_option_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonServerOptionPoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}
mt = TipcMtLookupByNum(T_MT_MON_SERVER_OPTION_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
                           process_mon_server_option_poll_result,
                           NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```


TipcMonServerRoutePoll

Name	TipcMonServerRoutePoll — poll once for RTserver route information
Synopsis	<pre>T_BOOL TipcMonServerRoutePoll(server_name, dest_server_name) T_STR server_name; T_STR dest_server_name;</pre>
Arguments	<i>server_name</i> — name of RTserver to poll for route information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope) <i>dest_server_name</i> — name of destination RTserver whose route in RTserver to poll (use T_IPC_MON_ALL to indicate a poll of routes to all other RTservers)
Return Values	TRUE if the poll for RTserver route information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonServerRoutePoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>server_name</i> or <i>dest_server_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonServerRoutePoll polls for RTserver route information by sending a MON_SERVER_ROUTE_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The route information to the RTserver <i>dest_server_name</i> is polled. The polled RTserver responds by sending back a MON_SERVER_ROUTE_POLL_RESULT message. The response should come back quickly. If <i>dest_server_name</i> is T_IPC_MON_ALL then the routes in the polled RTserver to all other RTservers are polled. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_SERVER_ROUTE_POLL_RESULT message contains one fixed field followed by one or more groups of four fields. The fixed field is a STR field containing the name of the RTserver. The group of four fields is: • a STR field containing the name of the destination RTserver • a STR_ARRAY field containing the names of the RTservers which are directly connected to the destination RTserver

- an INT4 field containing the distance (in connections or hops) from the polled RTserver to the destination RTserver
- a STR_ARRAY field containing the names of the RTservers to follow for the shortest path from the polled RTserver to the destination RTserver

There are multiple responses only if *server_name* is a wildcard or T_IPC_MON_ALL.

Caution TipcMonServerRoutePoll sends a MON_SERVER_ROUTE_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message routing.

There is no response message if the RTserver named *server_name* or destination RTserver named *dest_server_name* does not exist; the poll is silently dropped.

RTserver route information can only be polled, not watched.

See Also TipcMonServerConnPoll, TipcSrvMsgWrite

Examples This example uses TipcMonServerRoutePoll to poll for all route information in all RTservers, creates a callback to process the incoming MON_SERVER_ROUTE_POLL_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_route_poll_result -- process a MON_SERVER_ROUTE_POLL_RESULT
message */
void T_ENTRY process_mon_server_route_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR server_name;
    T_STR dest_server_name;
    T_STR *connected_server_names;
    T_INT4 num_connected_server_names;
    T_INT4 distance;
    T_STR *route_to_server;
    T_INT4 num_route_to_server;
    T_INT4 counter;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get first fixed field */
    if (!TipcMsgNextStr(data->msg, &server_name)) {
        return; /* error */
    }
    TutOut("Got SERVER_ROUTE poll result.\n");
    TutOut("RTserver name = %s\n", server_name);
```

```

/* loop through all groups */
while (TipcMsgRead(data->msg,
    T_IPC_FT_STR, &dest_server_name,
    T_IPC_FT_STR_ARRAY, &connected_server_names,
    &num_connected_server_names,
    T_IPC_FT_INT4, &distance,
    T_IPC_FT_STR_ARRAY, &route_to_server,
    &num_route_to_server,
    NULL)) {
    TutOut("destination server name = %s\n", dest_server_name);
    TutOut("connected servers:\n");
    for (counter = 0; counter < num_connected_server_names;
        counter++) {
        TutOut("  %s\n", connected_server_names[counter]);
    }
    TutOut("distance = %d\n", distance);
    TutOut("route:\n");
    for (counter = 0; counter < num_route_to_server;
        counter++) {
        TutOut("  %s\n", route_to_server[counter]);
    }
}

/* make sure we reached the end of the message */
if (TutErrNumGet() != T_ERR_MSG_EOM) {
    return; /* error */
}

} /* process_mon_server_route_poll_result */

/* ===== */
/*...code from calling function is below */
T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonServerRoutePoll("/...", T_IPC_MON_ALL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_ROUTE_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
    process_mon_server_route_poll_result,
    NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonServerStartTimePoll

Name	TipcMonServerStartTimePoll — poll once for RTserver start time and elapsed time
Synopsis	<pre>T_BOOL TipcMonServerStartTimePoll(server_name) T_STR server_name;</pre>
Arguments	<i>server_name</i> — name of RTserver to poll (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)
Return Values	TRUE if the poll was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonServerStartTimePoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>server_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonServerStartTimePoll polls for RTserver start time and elapsed time by sending a MON_SERVER_START_TIME_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTserver responds by sending back a MON_SERVER_START_TIME_POLL_RESULT message. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_SERVER_START_TIME_POLL_RESULT message contains three fields: • a STR field containing the name of the RTserver • a REAL8 field containing the start time (in seconds since 1970) of the RTserver • a REAL8 field containing the elapsed time (in seconds) of the RTserver There are multiple responses only if <i>server_name</i> is a wildcard or T_IPC_MON_ALL.

Caution TipcMonServerStartTimePoll sends a MON_SERVER_START_TIME_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no response message if the RTserver named *server_name* does not exist; the poll is silently dropped.

RTserver time information can be polled, but it cannot be watched.

See Also TipcMonServerTimePoll

Examples This example uses TipcMonServerStartTimePoll to poll for start time and elapsed time in all RTservers, creates a callback to process the incoming MON_SERVER_START_TIME_POLL_RESULT messages, then waits up to 10 seconds for the poll results:

```
void T_ENTRY ServerStartTimeCb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR server_name;
    T_REAL8 start_time;
    T_REAL8 elapsed;
    T_INT4 days = 0;
    T_INT4 hours = 0;
    T_INT4 minutes = 0;
    T_INT4 seconds = 0;

    TipcMsgNextStr(data->msg, &server_name);
    TipcMsgNextReal8(data->msg, &start_time);
    TipcMsgNextReal8(data->msg, &elapsed);

    if (elapsed > 86400.0 ) {
        days = (T_INT4)floor(elapsed / 86400);
        elapsed -= days*86400;
    }
    if (elapsed > 3600.0) {
        hours = (T_INT4)floor(elapsed / 3600);
        elapsed -= hours*3600;
    }
    if (elapsed > 60.0) {
        minutes = (T_INT4)floor(elapsed / 60);
        elapsed -= minutes*60;
    }
    seconds = elapsed;

    TutOut("RTserver was started %d days %d hours %d minutes %d
seconds ago\n",
        days, hours, minutes, seconds);
} /* ServerStartTimeCb */
```

```

/* ... code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_SERVER_START_TIME_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, ServerStartTimeCb, NULL) == NULL) {
    return; /* error */
}

TipcMonServerStartTimePoll("/...");
TipcSrvFlush();
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonServerTimePoll

Name	TipcMonServerTimePoll — poll once for RTserver time information
Synopsis	<pre>T_BOOL TipcMonServerTimePoll(server_name) T_STR server_name;</pre>
Arguments	<i>server_name</i> — name of RTserver to poll for time information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)
Return Values	TRUE if the poll for RTserver time information was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonServerTimePoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>server_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonServerTimePoll polls for RTserver time information by sending a MON_SERVER_TIME_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTserver responds by sending back a MON_SERVER_TIME_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_SERVER_TIME_POLL_RESULT message contains four fields: • a STR field containing the name of the RTserver • a REAL8 field containing the wall time (the time from TutGetWallTime) • a STR field containing the wall time converted to a string using the full time format • a REAL8 field containing the amount of CPU time used There are multiple responses only if <i>server_name</i> is a wildcard or T_IPC_MON_ALL. To poll once for RTclient time information, use TipcMonClientTimePoll.

Caution TipcMonServerTimePoll sends a MON_SERVER_TIME_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no response message if the RTserver named *server_name* does not exist; the poll is silently dropped.

RTserver time information can only be polled, not watched, as RTserver does not have the notion of current time as RTclients do.

See Also TipcMonClientTimePoll, TipcSrvMsgWrite

Examples This example uses TipcMonServerTimePoll to poll for time information in all RTservers, creates a callback to process the incoming MON_SERVER_TIME_POLL_RESULT messages, then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_time_poll_result -- process a MON_SERVER_TIME_POLL_RESULT
message */
void T_ENTRY process_mon_server_time_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR server_name;
    T_REAL8 wall_time;
    T_STR wall_time_str;
    T_REAL8 cpu_time;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgRead(data->msg,
                    T_IPC_FT_STR, &server_name,
                    T_IPC_FT_REAL8, &wall_time,
                    T_IPC_FT_STR, &wall_time_str,
                    T_IPC_FT_REAL8, &cpu_time,
                    NULL)) {
        return; /* error */
    }

    TutOut("Got SERVER_TIME poll result.\n");
    TutOut("RTserver name = %s\n", server_name);
    TutOut("wall time = %f\n", wall_time);
    TutOut("wall time str = %s\n", wall_time_str);
    TutOut("CPU time = %f\n", cpu_time);
} /* process_mon_server_time_poll_result */
```



```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcMonServerTimePoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_TIME_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
    process_mon_server_time_poll_result,
    NULL) == NULL) {
    return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}

```

TipcMonServerVersionPoll

Name	TipcMonServerVersionPoll — poll once for RTserver version information
Synopsis	<pre>T_BOOL TipcMonServerVersionPoll(server_name) T_STR server_name;</pre>
Arguments	<i>server_name</i> — name of RTserver to poll (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)
Return Values	TRUE if the poll was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonServerVersionPoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>server_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonServerVersionPoll polls for RTserver version information by sending a MON_SERVER_VERSION_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTserver responds by sending back a MON_SERVER_VERSION_POLL_RESULT message. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. Each MON_SERVER_VERSION_POLL_RESULT message contains two fields: • a STR field containing the name of the RTserver • a INT4 field containing the version of the RTserver There are multiple responses only if <i>server_name</i> is a wildcard or T_IPC_MON_ALL. Caution TipcMonServerVersionPoll sends a MON_SERVER_VERSION_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering. There is no response message if the RTserver named <i>server_name</i> does not exist; the poll is silently dropped. RTserver version information can be polled, but it cannot be watched.
See Also	TipcMonClientVersionPoll

Examples This example uses TipcMonServerVersionPoll to poll for version information in all RTservers, creates a callback to process the incoming MON_SERVER_VERSION_POLL_RESULT messages, then waits up to 10 seconds for the poll results:

```
void T_ENTRY ServerVersionCb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR server_name;
    T_INT4 version;

    TipcMsgNextStr(data->msg, &server_name);
    TipcMsgNextInt4(data->msg, &version);
    TutOut("%s: %d\n", server_name, version);
} /* ServerVersionCb */

/* ... code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_SERVER_VERSION_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, ServerVersionCb, NULL) == NULL) {
    return; /* error */
}

TipcMonServerVersionPoll("/...");
TipcSrvFlush();
if (!TipcSrvMainLoop(10.0)) {
    return; /* error */
}
```

TipcMonSetIdentStr

Name	TipcMonSetIdentStr — set the monitoring identification string of this process
Synopsis	<pre>T_BOOL TipcMonSetIdentStr(ident_str) T_STR ident_str;</pre>
Arguments	<i>ident_str</i> — identification string
Return Values	TRUE if the identification string was successfully set for this process, FALSE otherwise.
Diagnostics	If TipcMonSetIdentStr fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>ident_str</i> was null
Description	TipcMonSetIdentStr sets the monitoring identification string for this process. This identification string is used as a descriptive name for the process when it is being monitored. See TipcMonGetIdentStr for more information on monitoring identification strings. The identification string can also be set with the Monitor_Ident option. See the <i>TIBCO SmartSockets User's Guide</i> for more information.
Caution	Each RTclient sets the monitoring identification string in TipcMonSetIdentStr. This string is stored in RTserver and is only sent to RTserver when RTclient connects to RTserver. A process that calls TipcMonSetIdentStr after calling TipcSrvCreate will be identified incorrectly.
See Also	TipcMonGetIdentStr
Examples	This example sets the monitoring identification string for this process to "Acme Inc. Custom RTclient": <pre>if (!TipcMonSetIdentStr("Acme Inc. Custom RTclient")) { return; /* error */ }</pre>

TipcMonSubjectNamesGetWatch

Name	TipcMonSubjectNamesGetWatch — determine if this RTclient is watching subject names
Synopsis	<pre>T_BOOL TipcMonSubjectNamesGetWatch(<i>watch_status_return</i>) T_BOOL *<i>watch_status_return</i>;</pre>
Arguments	<i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the subject names watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonSubjectNamesGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>watch_status_return</i> was null
Description	TipcMonSubjectNamesGetWatch gets whether this RTclient is watching the subject names in the current project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_SUBJECT_NAMES_STATUS message each time a subject is created or destroyed. A subject is considered created when the first RTclient starts subscribing to that subject. A subject is considered destroyed when the last RTclient stops subscribing to that subject. Each MON_SUBJECT_NAMES_STATUS message contains three fields: • a STR_ARRAY field containing the current subject names • a STR field containing the name of the created subject (" " if not applicable) • a STR field containing the name of the destroyed subject (" " if not applicable) Use TipcMonSubjectNamesSetWatch to set whether this RTclient is watching subject names. To poll once for subject names, use TipcMonSubjectNamesPoll.
Caution	If TipcMonSubjectNamesGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i> .
See Also	TipcMonSubjectNamesPoll, TipcMonSubjectNamesSetWatch

Examples This uses TipcMonSubjectNamesGetWatch to get the status of watching subject names:

```
T_BOOL watch_status;
if (!TipcMonSubjectNamesGetWatch(&watch_status)) {
    return; /* error */
}
TutOut("This process is %s subject names.\n",
       watch_status ? "watching" : "not watching");
```

TipcMonSubjectNamesPoll

Name	TipcMonSubjectNamesPoll — poll once for subject names
Synopsis	<code>T_BOOL TipcMonSubjectNamesPoll()</code>
Arguments	None
Return Values	TRUE if the poll for subject names was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonSubjectNamesPoll fails, it returns FALSE and sets the global SmartSockets error number to: • any error number from TipcSrvMsgWrite
Description	TipcMonSubjectNamesPoll polls for the subject names in the current project that match the value of the Monitor_Scope option by sending a MON_SUBJECT_NAMES_POLL_CALL message to RTserver. RTserver responds by sending back a MON_SUBJECT_NAMES_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. The MON_SUBJECT_NAMES_POLL_RESULT message contains one field, a STR_ARRAY field containing the current subject names. To watch for created and destroyed subject names, use TipcMonSubjectNamesSetWatch.
Caution	TipcMonSubjectNamesPoll sends a MON_SUBJECT_NAMES_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.
See Also	TipcMonSubjectNamesGetWatch, TipcMonSubjectNamesSetWatch, TipcSrvMsgWrite

Examples This example uses TipcMonSubjectNamesPoll to poll for subject names, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *subject_names;
T_INT4 num_subject_names;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcMonSubjectNamesPoll()) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SUBJECT_NAMES_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvMsgSearchType(10.0, mt);
if (msg == NULL) {
    return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}

if (!TipcMsgNextStrArray(msg, &subject_names, &num_subject_names))
{
    return; /* error */
}

TutOut("Got SUBJECT_NAMES poll result.\n");
for (counter = 0; counter < num_subject_names; counter++) {
    TutOut("subject_names[%d] = %s\n", counter,
        subject_names[counter]);
}

/* clean up */
if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```


TipcMonSubjectNamesSetWatch

Name	TipcMonSubjectNamesSetWatch — start or stop watching subject names
Synopsis	<pre>T_BOOL TipcMonSubjectNamesSetWatch(<i>watch_status</i>) T_BOOL <i>watch_status</i>;</pre>
Arguments	<i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the subject names watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonSubjectNamesSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonSubjectNamesSetWatch sets whether this RTclient is watching the subject names in the current project that match the value of the Monitor_Scope option. TipcMonSubjectNamesSetWatch saves the watch status and sends a MON_SUBJECT_NAMES_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_SUBJECT_NAMES_STATUS message each time a subject is created or destroyed. A subject is considered created when the first RTclient starts subscribing to that subject. A subject is considered destroyed when the last RTclient stops subscribing to that subject. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. Each MON_SUBJECT_NAMES_STATUS message contains three fields: • a STR_ARRAY field containing the current subject names • a STR field containing the name of the created subject (or "" if not applicable) • a STR field containing the name of the destroyed subject (or "" if not applicable) In addition to sending a MON_SUBJECT_NAMES_STATUS message each time a subject is created or destroyed, RTserver also sends an initial MON_SUBJECT_NAMES_STATUS message when the watch status is set to TRUE, so that the RTclient does not have to wait for the next subject name change to receive the first MON_SUBJECT_NAMES_STATUS message. To determine if this RTclient is watching subject names, use TipcMonSubjectNamesGetWatch. To poll once for subject names, use TipcMonSubjectNamesPoll.

Caution TipcMonSubjectNamesSetWatch sends a MON_SUBJECT_NAMES_SET_WATCH message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

See Also TipcMonSubjectNamesGetWatch, TipcMonSubjectNamesPoll

Examples This example uses TipcMonSubjectNamesSetWatch to start watching subject names and creates a callback to process the incoming MON_SUBJECT_NAMES_STATUS messages:

```
/* ===== */
/*..process_mon_subject_names_status -- process a MON_SUBJECT_NAMES_STATUS message
*/
void T_ENTRY process_mon_subject_names_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR *subject_names;
    T_INT4 num_subject_names;
    T_STR created_subject;
    T_STR destroyed_subject;
    T_INT4 counter;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* get the fields from the message */
    if (!TipcMsgNextStrArray(data->msg, &subject_names,
                            &num_subject_names)
        || !TipcMsgNextStr(data->msg, &created_subject)
        || !TipcMsgNextStr(data->msg, &destroyed_subject)) {
        return; /* error */
    }

    TutOut("Got SUBJECT_NAMES status.\n");
    for (counter = 0; counter < num_subject_names; counter++) {
        TutOut("subject_names[%d] = %s\n", counter,
              subject_names[counter]);
    }
    TutOut("created subject = %s\n", created_subject);
    TutOut("destroyed subject = %s\n", destroyed_subject);
} /* process_mon_subject_names_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;
if (!TipcMonSubjectNamesSetWatch(TRUE)) {
    return; /* error */
}
mt = TipcMtLookupByNum(T_MT_MON_SUBJECT_NAMES_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_mon_subject_names_status,
NULL)
    == NULL) {
    return; /* error */
}
/* At this point TipcSrvMainLoop can be used to read and process messages. */

```

TipcMonSubjectSubscribeGetWatch

Name	TipcMonSubjectSubscribeGetWatch — determine if this RTclient is watching the RTclients that are subscribing to a subject
Synopsis	<pre>T_BOOL TipcMonSubjectSubscribeGetWatch(subject_name, watch_status_return) T_STR subject_name; T_BOOL *watch_status_return;</pre>
Arguments	<i>subject_name</i> — name of subject to get watch status for (use T_IPC_MON_ALL to indicate a watch of all subjects matching the value of Monitor_Scope) <i>watch_status_return</i> — storage for watch status
Return Values	TRUE if the subject subscribe watch status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcMonSubjectSubscribeGetWatch fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>subject_name</i> or <i>watch_status_return</i> was null • T_ERR_VAL_INVALID — <i>subject_name</i> was not a valid subject
Description	TipcMonSubjectSubscribeGetWatch gets whether this RTclient is watching the RTclients that are subscribing to a subject. The watch status controls whether the RTclient receives a MON_SUBJECT_SUBSCRIBE_STATUS message each time an RTclient starts or stops subscribing to the subject. If <i>subject_name</i> is T_IPC_MON_ALL then all subjects in the current project that match the value of the Monitor_Scope option are watched. Each MON_SUBJECT_SUBSCRIBE_STATUS message contains four fields: • a STR field containing the name of the subject • a STR_ARRAY field containing the names of the current RTclients that are subscribing to the subject • a STR field containing the name of the RTclient that just started subscribing to the subject (or "" if not applicable) • a STR field containing the name of the RTclient that just stopped subscribing to the subject (or "" if not applicable) To set whether this RTclient is watching the RTclients that are subscribing to a subject, use TipcMonSubjectSubscribeSetWatch. To poll once for the RTclients that are subscribing to a subject, use TipcMonSubjectSubscribePoll.

For an example of how MON_SUBJECT_SUBSCRIBE_STATUS messages can be used, see the RTmon GDI Watch Clients in Subject window.

Caution If TipcMonSubjectSubscribeGetWatch returns FALSE, it does not store a value in *watch_status_return*.

See Also TipcMonSubjectSubscribePoll, TipcMonSubjectSubscribeSetWatch

Examples This example uses TipcMonSubjectSubscribeGetWatch to get the status of watching the RTclients that are subscribing to the thermal subject:

```
T_BOOL watch_status;

if (!TipcMonSubjectSubscribeGetWatch("thermal", &watch_status)) {
    return; /* error */
}
TutOut("This process is %s subject subscribers for thermal.\n",
       watch_status ? "watching" : "not watching");
```

TipcMonSubjectSubscribePoll

Name	TipcMonSubjectSubscribePoll — poll once for the RTclients that are subscribing to a subject
Synopsis	<pre>T_BOOL TipcMonSubjectSubscribePoll(<i>subject_name</i>) T_STR <i>subject_name</i>;</pre>
Arguments	<i>subject_name</i> — name of subject to poll for subscribers (use T_IPC_MON_ALL to indicate a poll of all subjects matching the value of Monitor_Scope)
Return Values	TRUE if the poll for the RTclients that are subscribing to a subject was successfully initiated, FALSE otherwise.
Diagnostics	If TipcMonSubjectSubscribePoll fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>subject_name</i> was null • any error number from TipcSrvMsgWrite
Description	TipcMonSubjectSubscribePoll polls for RTclients that are subscribing to a subject by sending a MON_SUBJECT_SUBSCRIBE_POLL_CALL message to RTserver. If <i>subject_name</i> is T_IPC_MON_ALL then all subjects in the current project that match the value of the Monitor_Scope option are polled. RTserver responds by sending back a MON_SUBJECT_SUBSCRIBE_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvMsgSearchType or TipcSrvMainLoop to get the response. The MON_SUBJECT_SUBSCRIBE_POLL_RESULT message contains groups of two fields: • a STR field containing the name of the subject • a STR_ARRAY field containing the names of the current RTclients that are subscribing to the subject The group of two fields is repeated multiple times only if <i>subject_name</i> is T_IPC_MON_ALL. To watch for subject subscribers, use TipcMonSubjectSubscribeSetWatch.
Caution	TipcMonSubjectSubscribePoll sends a MON_SUBJECT_SUBSCRIBE_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

See Also TipcMonSubjectSubscribeGetWatch, TipcMonSubjectSubscribeSetWatch, TipcSrvMsgWrite

Examples This example uses TipcMonSubjectSubscribePoll to poll all subjects for the RTclients that are subscribing to the subject, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR subject_name;
T_STR *subscribe_client_names;
T_INT4 num_subscribe_client_names;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcMonSubjectSubscribePoll("/...")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SUBJECT_SUBSCRIBE_POLL_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvMsgSearchType(10.0, mt);
if (msg == NULL) {
    return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}

/* loop through all groups */
TutOut("Got SUBJECT_SUBSCRIBE poll result.\n");
while (TipcMsgNextStr(msg, &subject_name)
      && TipcMsgNextStrArray(msg, &subscribe_client_names,
                             &num_subscribe_client_names)) {
    TutOut("subject name = %s\n", subject_name);
    for (counter = 0; counter < num_subscribe_client_names;
         counter++) {
        TutOut("subscribe_client_names[%d] = %s\n", counter,
               subscribe_client_names[counter]);
    }
}
```

```
/* make sure we reached the end of the message */  
if (TutErrNumGet() != T_ERR_MSG_EOM) {  
    return; /* error */  
}  
  
/* clean up */  
if (!TipcMsgDestroy(msg)) {  
    return; /* error */  
}
```


TipcMonSubjectSubscribeSetWatch

Name	TipcMonSubjectSubscribeSetWatch — start or stop watching the RTclients that are subscribing to a subject
Synopsis	<pre>T_BOOL TipcMonSubjectSubscribeSetWatch(subject_name, watch_status) T_STR subject_name; T_BOOL watch_status;</pre>
Arguments	<i>subject_name</i> — name of subject to set watch status for (use T_IPC_MON_ALL to indicate a watch of all subjects matching the value of Monitor_Scope) <i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)
Return Values	TRUE if the subject subscribe watch status was successfully set, FALSE otherwise.
Diagnostics	If TipcMonSubjectSubscribeSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>subject_name</i> was null • T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE, or <i>subject_name</i> was not a valid subject • T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status
Description	TipcMonSubjectSubscribeSetWatch sets whether this RTclient is watching the RTclients that are subscribing to a subject. TipcMonSubjectSubscribeSetWatch saves the watch status and sends a MON_SUBJECT_SUBSCRIBE_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_SUBJECT_SUBSCRIBE_STATUS message each time an RTclient starts or stops subscribing to the subject. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. If <i>subject_name</i> is T_IPC_MON_ALL then all subjects in the current project that match the value of the Monitor_Scope option are watched. Each MON_SUBJECT_SUBSCRIBE_STATUS message contains four fields: • a STR field containing the name of the subject • a STR_ARRAY field containing the names of the current RTclients that are subscribing to the subject • a STR field containing the name of the RTclient that just started subscribing to the subject (or "" if not applicable) • a STR field containing the name of the RTclient that just stopped subscribing to the subject (or "" if not applicable)

In addition to sending a `MON_SUBJECT_SUBSCRIBE_STATUS` message each time an RTclient starts or stops subscribing to the subject, RTserver also sends an initial `MON_SUBJECT_SUBSCRIBE_STATUS` message (or many messages if `T_IPC_MON_ALL` is used) when the watch status is set to `TRUE`, so that the RTclient does not have to wait for the next subject subscriber change to receive the first `MON_SUBJECT_SUBSCRIBE_STATUS` message.

To determine if this RTclient is watching the RTclients that are subscribing to a subject, use `TipcMonSubjectSubscribeGetWatch`. To poll once for the RTclients that are subscribing to a subject, use `TipcMonSubjectSubscribePoll`.

For an example of how `MON_SUBJECT_SUBSCRIBE_STATUS` messages can be used, see the RTmon GDI Watch Clients in Subject window.

Caution `TipcMonSubjectSubscribeSetWatch` sends a `MON_SUBJECT_SUBSCRIBE_SET_WATCH` message to RTserver, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no initial status message if the subject named *subject_name* does not exist.

See Also `TipcMonSubjectSubscribeGetWatch`, `TipcMonSubjectSubscribePoll`

Examples This example uses `TipcMonSubjectSubscribeSetWatch` to start watching the RTclients that are subscribing to the thermal subject and creates a callback to process the incoming `MON_SUBJECT_SUBSCRIBE_STATUS` messages:

```
/* ===== */
/*..process_mon_subject_subscribe_status -- process a MON_SUBJECT_SUBSCRIBE_STATUS
message */
void T_ENTRY process_mon_subject_subscribe_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR subject_name;
    T_STR *subscribe_client_names;
    T_INT4 num_subscribe_client_names;
    T_STR start_client;
    T_STR stop_client;
    T_INT4 counter;

    /* must set current field first */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }
}
```

```

/* get the fields from the message */
if (!TipcMsgNextStr(data->msg, &subject_name)
    || !TipcMsgNextStrArray(data->msg, &subscribe_client_names,
                             &num_subscribe_client_names)
    || !TipcMsgNextStr(data->msg, &start_client)
    || !TipcMsgNextStr(data->msg, &stop_client)) {
    return; /* error */
}

TutOut("Got SUBJECT_SUBSCRIBE status.\n");
TutOut("subject name = %s\n", subject_name);
for (counter = 0; counter < num_subscribe_client_names;
counter++) {
    TutOut("subscribe_client_names[%d] = %s\n", counter,
           subscribe_client_names[counter]);
}
TutOut("start RTclient = %s\n", start_client);
TutOut("stop RTclient = %s\n", stop_client);
} /* process_mon_subject_subscribe_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcMonSubjectSubscribeSetWatch("thermal", TRUE)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SUBJECT_SUBSCRIBE_STATUS);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt,
process_mon_subject_subscribe_status,
                           NULL) == NULL) {
    return; /* error */
}

/* At this point TipcSrvMainLoop can be used to read and process messages. */

```


TipcMsgAck - TipcMsgAddNamedXmlPtr

This chapter covers these TipcMsg API functions:

- *TipcMsgAck*
- *TipcMsgAddNamedBinary*
- *TipcMsgAddNamedBinaryPtr*
- *TipcMsgAddNamedBool*
- *TipcMsgAddNamedBoolArray*
- *TipcMsgAddNamedBoolArrayPtr*
- *TipcMsgAddNamedByte*
- *TipcMsgAddNamedChar*
- *TipcMsgAddNamedInt2*
- *TipcMsgAddNamedInt2Array*
- *TipcMsgAddNamedInt2ArrayPtr*
- *TipcMsgAddNamedInt4*
- *TipcMsgAddNamedInt4Array*
- *TipcMsgAddNamedInt4ArrayPtr*
- *TipcMsgAddNamedInt8*
- *TipcMsgAddNamedInt8Array*
- *TipcMsgAddNamedInt8ArrayPtr*
- *TipcMsgAddNamedMsg*
- *TipcMsgAddNamedMsgArray*
- *TipcMsgAddNamedMsgArrayPtr*
- *TipcMsgAddNamedMsgPtr*
- *TipcMsgAddNamedReal4*
- *TipcMsgAddNamedReal4Array*
- *TipcMsgAddNamedReal4ArrayPtr*
- *TipcMsgAddNamedReal8*

- *TpcMsgAddNamedReal8Array*
- *TpcMsgAddNamedReal8ArrayPtr*
- *TpcMsgAddNamedReal16*
- *TpcMsgAddNamedReal16Array*
- *TpcMsgAddNamedReal16ArrayPtr*
- *TpcMsgAddNamedStr*
- *TpcMsgAddNamedStrArray*
- *TpcMsgAddNamedStrArrayPtr*
- *TpcMsgAddNamedStrPtr*
- *TpcMsgAddNamedTimestamp*
- *TpcMsgAddNamedTimestampArray*
- *TpcMsgAddNamedTimestampArrayPtr*
- *TpcMsgAddNamedUnknown*
- *TpcMsgAddNamedUtf8*
- *TpcMsgAddNamedUtf8Array*
- *TpcMsgAddNamedUtf8ArrayPtr*
- *TpcMsgAddNamedUtf8Ptr*
- *TpcMsgAddNamedXml*
- *TpcMsgAddNamedXmlPtr*

TipcMsgAck

Name	TipcMsgAck — acknowledge delivery of a message
Synopsis	<pre>T_BOOL TipcMsgAck(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to acknowledge
Return Values	TRUE if the message was successfully acknowledged, FALSE otherwise.
Diagnostics	If TipcMsgAck fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_TYPE_INVALID — <i>msg</i> does not need to be acknowledged • any error number from TipcConnMsgSend
Description	TipcMsgAck acknowledges successful delivery of a message by a receiving process. When a message with a delivery mode of T_IPC_DELIVERY_SOME or T_IPC_DELIVERY_ALL is sent through a connection with TipcConnMsgSend, the sender saves a copy of the message until some (in the case of T_IPC_DELIVERY_SOME) or all (in the case of T_IPC_DELIVERY_ALL) receiving processes acknowledge that they have received the message. TipcMsgDestroy automatically calls TipcMsgAck to ensure a received message is acknowledged before it is destroyed. TipcMsgDestroy assumes that when a receiving process destroys a message, it is finished processing the message and that the message can be considered to have been successfully delivered to this receiving process. TipcMsgAck first updates the highest sequence number (based on the sequence number and sender of <i>msg</i>) in the GMD area of the connection the message was received on. The highest sequence number is used to detect duplicate messages. TipcMsgAck then creates a GMD_ACK message, appends an INT4 integer field containing the sequence number of the message being acknowledged, and sends the GMD_ACK message with TipcConnMsgSend through the same connection the message was received on.

If the message's delivery mode is either `T_IPC_DELIVERY_SOME` or `T_IPC_DELIVERY_ALL` and the message has not already been acknowledged manually with `TipcMsgAck`, then `TipcMsgDestroy` first calls `TipcMsgAck` to acknowledge successful delivery of the message to this receiving process.

Caution `TipcMsgAck` is a low-level function that is normally not used directly by developers, but called by `TipcMsgDestroy`. Normally, `TipcMsgAck` is used only when a process receives a message, processes it, and then needs to keep the message around for any period of time. When the receiver uses `TipcMsgAck` in this case it allows the sending process to destroy its own saved copy of the message.

`TipcMsgAck` sends a `GMD_ACK` message, but does not explicitly flush the message. See `TipcConnGetAutoFlushSize` for more information on message buffering.

See Also `TipcMsgCreate`, `TipcMsgDestroy`, `TipcConnMsgSend`

Examples This example reads a message from a connection, processes it, and manually acknowledges it with `TipcMsgAck`:

```
T_IPC_MSG msg;

msg = TipcConnMsgNext(conn, T_TIMEOUT_FOREVER);
if (msg == NULL) {
    return; /* error */
}

/* Process the message however appropriate: either by using TipcConnMsgProcess to call
process/default callbacks, or by using TipcMsgNext* to directly access the fields of the message. */

/* Acknowledge delivery of the message. You should only do this if you plan to keep the message
around for a while. */
if (!TipcMsgAck(msg)) {
    /* T_ERR_TYPE_INVALID in this case just means no ack needed. */
    if (TutErrNumGet() != T_ERR_TYPE_INVALID) {
        return; /* error */
    }
}
```


TipcMsgAddNamedBinary

Name	TipcMsgAddNamedBinary — add a BINARY field to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedBinary(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_PTR value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — size of the value, in bytes
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedBinary fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedBinary adds a BINARY field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendBinary, TipcMsgUpdateNamedBinary, TipcMsgGetNamedBinary
Examples	This example creates a message and adds an existing BINARY value to it: <pre>T_IPC_MSG msg; T_BOOL status; T_PTR bin_value; T_INT4 size; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } status = TipcMsgAddNamedBinary(msg, "foo", bin_value, size); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedBinaryPtr

Name	TipcMsgAddNamedBinaryPtr — add a BINARY field to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedBinaryPtr(msg, name, ptr, size, field_return) T_IPC_MSG msg; T_STR name; T_PTR ptr; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>ptr</i> — pointer to the value of the field <i>size</i> — number of bytes in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedBinaryPtr fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedBinaryPtr adds a BINARY field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendBinaryPtr, TipcMsgUpdateNamedBinaryPtr, TipcMsgFieldUpdateBinaryPtr

Examples This example creates a message and adds a BINARY value to it using a pointer:

```
T_IPC_MSG msg;
T_BOOL status;
T_PTR ptr;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

status = TipcMsgAddNamedBinaryPtr(msg, "foo", ptr, size,
field_return);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedBool

Name TipcMsgAddNamedBool — add a boolean field to a message using a name

Synopsis `T_BOOL TipcMsgAddNamedBool(msg, name)`
`T_IPC_MSG msg;`
`T_STR name;`
`T_BOOL value;`

Arguments *msg* — message to add field to
name — name of the field
value — value of the field

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedBool fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedBool adds a BOOL field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.

See Also TipcMsgAppendBool, TipcMsgUpdateNamedBool, TipcMsgGetNamedBool

Examples This example creates a message and adds a BOOL value to it:

```
T_IPC_MSG msg;
T_BOOL status;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

status = TipcMsgAddNamedBool(msg, "foo", T_TRUE);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedBoolArray

Name	TipcMsgAddNamedBoolArray — add a field containing an array of BOOL fields to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedBoolArray(msg, name, size) T_IPC_MSG msg; T_STR name; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>size</i> — number of bytes in the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedBoolArray fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedBoolArray adds a BOOL_ARRAY field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendBoolArray, TipcMsgUpdateNamedBoolArray, TipcMsgGetNamedBoolArray
Examples	This example creates a message and adds a BOOL_ARRAY value to it: <pre>T_IPC_MSG msg; T_BOOL status; T_BOOL *array; T_INT4 array_size; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } array_size = 2; array[0] = T_TRUE; array[1] = T_FALSE; status = TipcMsgAddNamedBoolArray(msg, "foo", array, array_size); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedBoolArrayPtr

Name TipcMsgAddNamedBoolArrayPtr — add a field containing an array of BOOL fields to a message using a name, and a pointer to the field value, rather than a copy

Synopsis `T_BOOL TipcMsgAddNamedBoolArrayPtr(msg, name, ptr, size, field_return)`
 `T_IPC_MSG msg;`
 `T_STR name;`
 `T_PTR ptr;`
 `T_INT4 size;`
 `T_IPC_MSG_FIELD *field_return;`

Arguments *msg* — message to add field to
 name — name of the field
 ptr — pointer to the value of the field
 size — number of bytes in the field
 field_return — optional value to return field (use NULL if not needed)

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedBoolArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedBoolArrayPtr adds a BOOL_ARRAY field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.

See Also TipcMsgAppendBoolArrayPtr, TipcMsgUpdateNamedBoolArrayPtr, TipcMsgFieldUpdateBoolArrayPtr

Examples This example creates a message and adds a BOOL_ARRAY value to it:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_BOOL data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_BOOL_ARRAY_TEST 100
mt = TipcMtCreate("bool_array_test", USER_MT_BOOL_ARRAY_TEST,
                 "bool_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedBoolArrayPtr(msg, "named_bool_array", data, 0,
&field)) {
    return; /* error */
}

data[0] = T_TRUE;
data[1] = T_FALSE;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}
```

TipcMsgAddNamedByte

Name	TipcMsgAddNamedByte — add a BYTE field to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedByte(msg, name, value) T_IPC_MSG msg; T_STR name; T_UCHAR value;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedByte fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedByte adds a BYTE field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendByte, TipcMsgUpdateNamedByte, TipcMsgGetNamedByte
Examples	This example creates a message and adds a BYTE value to it using a pointer: <pre>T_IPC_MSG msg; T_BOOL status; T_UCHAR data; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } data = 0xFF; status = TipcMsgAddNamedByte(msg, "foo", data);</pre>

TipcMsgAddNamedChar

Name	TipcMsgAddNamedChar — add a character field to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedChar(msg, name, value) T_IPC_MSG msg; T_STR name; T_CHAR value;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedChar fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedChar adds a character field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendChar, TipcMsgUpdateNamedChar, TipcMsgGetNamedChar
Examples	This example creates a message and adds a character value to it: <pre>T_IPC_MSG msg; T_BOOL status; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } status = TipcMsgAddNamedChar(msg, "foo", 'c'); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedInt2

Name	TipcMsgAddNamedInt2 — add an INT2 field to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedInt2(msg, name, value) T_IPC_MSG msg; T_STR name; T_INT2 value;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedInt2 fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedInt2 adds an INT2 field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendInt2, TipcMsgUpdateNamedInt2, TipcMsgGetNamedInt2
Examples	This example creates a message and adds an INT2 value to it: <pre>T_IPC_MSG msg; T_BOOL status; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } status = TipcMsgAddNamedInt2(msg, "foo", (T_INT2)5); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedInt2Array

Name	TipcMsgAddNamedInt2Array — add a field containing an array of INT2 fields to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedInt2Array(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_INT2 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedInt2Array fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedInt2Array adds a field containing an array of INT2 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendInt2Array, TipcMsgUpdateNamedInt2Array, TipcMsgGetNamedInt2Array

Examples This example creates a message and adds an INT2_ARRAY value to it:

```
T_IPC_MSG msg;
T_BOOL status;
T_INT2 *array;
T_INT4 array_size;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

array_size = 2;
array[0] = 56;
array[1] = 57;

status = TipcMsgAddNamedInt2Array(msg, "foo", array, array_size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedInt2ArrayPtr

Name	TipcMsgAddNamedInt2ArrayPtr — add a field containing an array of INT2 fields to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedInt2ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_INT2 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedInt2ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedInt2ArrayPtr adds a field containing an array of INT2 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendInt2ArrayPtr, TipcMsgUpdateNamedInt2ArrayPtr, TipcMsgFieldUpdateInt2ArrayPtr

Examples This example creates a message and adds a pointer to an INT2_ARRAY value to it:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT2 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_INT2_ARRAY_TEST 100
mt = TipcMtCreate("int2_array_test", USER_MT_INT2_ARRAY_TEST,
                 "int2_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedInt2ArrayPtr(msg, "named_int2_array", data, 0,
                                &field)) {
    return; /* error */
}

data[0] = 56;
data[1] = 57;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}
```

TipcMsgAddNamedInt4

Name	TipcMsgAddNamedInt4 — add an INT4 field to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedInt4(msg, name, value) T_IPC_MSG msg; T_STR name; T_INT4 value;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedInt4 fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedInt4 adds an INT4 field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendInt4, TipcMsgUpdateNamedInt4, TipcMsgGetNamedInt4
Examples	This example creates a message and adds an INT4 value to it: <pre>T_IPC_MSG msg; T_BOOL status; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } status = TipcMsgAddNamedInt4(msg, "foo", (T_INT4)5); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedInt4Array

Name TipcMsgAddNamedInt4Array — add a field containing an array of INT4 fields to a message using a name

Synopsis

```
T_BOOL TipcMsgAddNamedInt4Array(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_INT4 *value;
T_INT4 size;
```

Arguments *msg* — message to add field to
name — name of the field
value — value of the field
size — number of elements in the field

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedInt4Array fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedInt4Array adds a field containing an array of INT4 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.

See Also TipcMsgAppendInt4Array, TipcMsgUpdateNamedInt4Array, TipcMsgGetNamedInt4Array

Examples This example creates a message and adds an INT4_ARRAY value to it:

```
T_IPC_MSG msg;
T_BOOL status;
T_INT4 *array;
T_INT4 array_size;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

array_size = 2;
array[0] = 560000;
array[1] = 570000;

status = TipcMsgAddNamedInt4Array(msg, "foo", array, array_size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedInt4ArrayPtr

Name	TipcMsgAddNamedInt4ArrayPtr — add a field containing an array of INT4 fields to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedInt4ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_INT4 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedInt4ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedInt4ArrayPtr adds a field containing an array of INT4 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendInt4ArrayPtr, TipcMsgUpdateNamedInt4ArrayPtr, TipcMsgFieldUpdateInt4ArrayPtr

Examples This example creates a message and adds an INT4_ARRAY value to it:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_INT4 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_INT4_ARRAY_TEST 100
mt = TipcMtCreate("int4_array_test", USER_MT_INT4_ARRAY_TEST,
                 "int4_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedInt4ArrayPtr(msg, "named_int4_array", data, 0,
                                &field)) {
    return; /* error */
}

data[0] = 560000;
data[1] = 570000;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

```

TipcMsgAddNamedInt8

Name TipcMsgAddNamedInt8 — add an INT8 field to a message using a name

Synopsis `T_BOOL TipcMsgAddNamedInt8(msg, name, value)`
`T_IPC_MSG msg;`
`T_STR name;`
`T_INT8 value;`

Arguments *msg* — message to add field to
name — name of the field
value — value of the field

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedInt8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedInt8 adds an INT8 field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.

See Also TipcMsgAppendInt8, TipcMsgUpdateNamedInt8, TipcMsgGetNamedInt8

Examples This example creates a message and adds an INT8 value to it:

```
T_IPC_MSG msg;
T_BOOL status;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

status = TipcMsgAddNamedInt8(msg, "foo", (T_INT8)5);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedInt8Array

Name	TipcMsgAddNamedInt8Array — add a field containing an array of INT8 fields to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedInt8Array(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_INT8 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedInt8Array fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedInt8Array adds a field containing an array of INT8 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendInt8Array, TipcMsgUpdateNamedInt8Array, TipcMsgGetNamedInt8Array

Examples This example creates a message and adds an INT8_ARRAY value to it:

```
T_IPC_MSG msg;
T_BOOL status;
T_INT8 *array;
T_INT4 array_size;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

array_size = 2;
array[0] = 56000000000;
array[1] = 57000000000;

status = TipcMsgAddNamedInt8Array(msg, "foo", array, array_size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedInt8ArrayPtr

Name	TipcMsgAddNamedInt8ArrayPtr — add a field containing an array of INT8 fields to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedInt8ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_INT8 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedInt8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedInt8ArrayPtr adds a field containing an array of INT8 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendInt8ArrayPtr, TipcMsgUpdateNamedInt8ArrayPtr, TipcMsgFieldUpdateInt8ArrayPtr

Examples This example creates a message and adds an INT8_ARRAY value to it:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT8 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_INT8_ARRAY_TEST 100
mt = TipcMtCreate("int8_array_test", USER_MT_INT8_ARRAY_TEST,
                 "int8_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedInt8ArrayPtr(msg, "named_int8_array", data, 0,
                                &field)) {
    return; /* error */
}

data[0] = 56000000000;
data[1] = 57000000000;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}
```


TipcMsgAddNamedMsg

Name	TipcMsgAddNamedMsg — add a MSG field to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedMsg(msg, name, value) T_IPC_MSG msg; T_STR name; T_IPC_MSG value;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedMsg fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedMsg adds a MSG field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendMsg, TipcMsgUpdateNamedMsg, TipcMsgGetNamedMsg
Examples	This example creates a message and adds a MSG field to it: <pre>T_IPC_MSG msg; T_IPC_MSG sub_msg; T_BOOL status; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } sub_msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_ALERT)); if (sub_msg == NULL) { return; /* error */ } status = TipcMsgAddNamedMsg(msg, "foo", sub_msg); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedMsgArray

Name	TipcMsgAddNamedMsgArray — add a field containing an array of MSG fields to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedMsgArray(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_MSG *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedMsgArray fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedMsgArray adds a field containing an array of MSG fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendMsgArray, TipcMsgUpdateNamedMsgArray, TipcMsgGetNamedMsgArray
Examples	This example creates a message and adds a MSG_ARRAY value to it: <pre>T_IPC_MT mt; T_IPC_MSG msg; #define ARRAY_SIZE 3 T_IPC_MSG msg_array_data[ARRAY_SIZE]; #define USER_MT_CONTAINER 3 mt = TipcMtCreate("container", USER_MT_CONTAINER, "msg_array"); if (mt == NULL) { return; /* error */ } msg = TipcMsgCreate(mt); if (msg == NULL) { return; /* error */ }</pre>

```

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

mt = TipcMtLookupByNum(T_MT_TIME);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[0] = TipcMsgCreate(mt);
if (msg_array_data[0] == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[1] = TipcMsgCreate(mt);
if (msg_array_data[1] == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_END_OF_FRAME);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[2] = TipcMsgCreate(mt);
if (msg_array_data[2] == NULL) {
    return; /* error */
}

if (!TipcMsgAppendReal8(msg_array_data[0], TutGetWallTime())) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg_array_data[1], "speed_limit",
55.0)) {
    return; /* error */
}

if (!TipcMsgAddNamedMsgArray(msg, "foo", msg_array_data,
ARRAY_SIZE)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```

TipcMsgAddNamedMsgArrayPtr

Name TipcMsgAddNamedMsgArrayPtr — add a field containing an array of MSG fields to a message using a name, and a pointer to the field value, rather than a copy

Synopsis `T_BOOL TipcMsgAddNamedMsgArrayPtr(msg, name, value, size, field_return)`
`T_IPC_MSG msg;`
`T_STR name;`
`T_MSG *value;`
`T_INT4 size;`
`T_IPC_MSG_FIELD *field_return;`

Arguments *msg* — message to add field to
name — name of the field
value — value of the field
size — number of elements in the field
field_return — optional value to return field (use NULL if not needed)

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedMsgArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedMsgArrayPtr adds a field containing an array of MSG fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.

See Also TipcMsgAppendMsgArrayPtr, TipcMsgUpdateNamedMsgArrayPtr, TipcMsgFieldUpdateMsgArrayPtr

Examples This example creates a message and adds a pointer to a MSG_ARRAY value to it:

```
T_IPC_MT mt;  
T_IPC_MSG msg;  
#define ARRAY_SIZE 3  
T_IPC_MSG msg_array_data[ARRAY_SIZE];
```

```

#define USER_MT_CONTAINER 3
mt = TipcMtCreate("container", USER_MT_CONTAINER, "msg_array");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_TIME);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[0] = TipcMsgCreate(mt);
if (msg_array_data[0] == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[1] = TipcMsgCreate(mt);
if (msg_array_data[1] == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_END_OF_FRAME);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[2] = TipcMsgCreate(mt);
if (msg_array_data[2] == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedMsgArrayPtr(msg, "named_msg_array",
                                msg_array_data,
                                ARRAY_SIZE, NULL)) {
    return; /* error */
}

if (!TipcMsgAppendReal8(msg_array_data[0], TutGetWallTime())) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg_array_data[1], "speed_limit",
70.0)) {
    return; /* error */
}

```

TipcMsgAddNamedMsgPtr

Name TipcMsgAddNamedMsgPtr — add a MSG field to a message using a name, and a pointer to the field value, rather than a copy

Synopsis `T_BOOL TipcMsgAddNamedMsgPtr(msg, name, ptr, field_return)`
`T_IPC_MSG msg;`
`T_STR name;`
`T_IPC_MSG ptr;`
`T_IPC_MSG_FIELD *field_return;`

Arguments *msg* — message to add field to
name — name of the field
ptr — pointer to the value of the field
field_return — optional value to return field (use NULL if not needed)

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedMsgPtr fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedMsgPtr adds a MSG field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.

See Also TipcMsgAppendMsgPtr, TipcMsgUpdateNamedMsgPtr, TipcMsgFieldUpdateMsgPtr

Examples This example creates a message and adds a MSG value to it using a pointer:

```
T_IPC_MSG msg;
T_BOOL status;
T_IPC_MSG data;
T_IPC_MSG_FIELD *field_return;
msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}
data = TipcMsgCreate(TipcMtLookupByNum(T_MT_NUMERIC_DATA));
status = TipcMsgAddNamedMsgPtr(msg, "foo", data, &field_return);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedReal4

Name	TipcMsgAddNamedReal4 — add a REAL4 field to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedReal4(msg, name, value) T_IPC_MSG msg; T_STR name; T_REAL4 value;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedReal4 fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedReal4 adds a REAL4 field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendReal4, TipcMsgUpdateNamedReal4, TipcMsgGetNamedReal4
Examples	This example creates a message and adds a REAL4 value to it: <pre>T_IPC_MSG msg; T_BOOL status; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } status = TipcMsgAddNamedReal4(msg, "foo", (T_REAL4)5.0); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedReal4Array

Name	TipcMsgAddNamedReal4Array — add a field containing an array of REAL4 fields to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedReal4Array(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_REAL4 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedReal4Array fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedReal4Array adds a field containing an array of REAL4 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendReal4Array, TipcMsgUpdateNamedReal4Array, TipcMsgGetNamedReal4Array

Examples This example creates a message and adds a REAL4_ARRAY value to it:

```
T_IPC_MSG msg;
T_BOOL status;
T_REAL4 *array;
T_INT4 array_size;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

array_size = 2;
array[0] = 56.1;
array[1] = 57.2;

status = TipcMsgAddNamedReal4Array(msg, "foo", array, array_size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedReal4ArrayPtr

Name TipcMsgAddNamedReal4ArrayPtr — add a field containing an array of REAL4 fields to a message using a name, and a pointer to the field value, rather than a copy

Synopsis T_BOOL TipcMsgAddNamedReal4ArrayPtr(*msg*, *name*, *value*, *size*, *field_return*)
 T_IPC_MSG *msg*;
 T_STR *name*;
 T_REAL4 **value*;
 T_INT4 *size*;
 T_IPC_MSG_FIELD **field_return*;

Arguments *msg* — message to add field to
 name — name of the field
 value — value of the field
 size — number of elements in the field
 field_return — optional value to return field (use NULL if not needed)

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedReal4ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedReal4ArrayPtr adds a field containing an array of REAL4 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.

See Also TipcMsgAppendReal4ArrayPtr, TipcMsgUpdateNamedReal4ArrayPtr, TipcMsgFieldUpdateReal4ArrayPtr

Examples This example creates a message and adds a REAL4_ARRAY value to it:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL4 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_REAL4_ARRAY_TEST 100
mt = TipcMtCreate("real4_array_test", USER_MT_REAL4_ARRAY_TEST,
                 "real4_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedReal4ArrayPtr(msg, "named_real4_array", data,
0,
                                &field)) {
    return; /* error */
}

data[0] = 56.1;
data[1] = 57.2;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

```

TipcMsgAddNamedReal8

Name	TipcMsgAddNamedReal8 — add a REAL8 field to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedReal8(msg, name, value) T_IPC_MSG msg; T_STR name; T_REAL8 value;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedReal8 fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedReal8 adds a REAL8 field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendReal8, TipcMsgUpdateNamedReal8, TipcMsgGetNamedReal8
Examples	This example creates a message and adds a REAL8 value to it: <pre>T_IPC_MSG msg; T_BOOL status; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } status = TipcMsgAddNamedReal8(msg, "foo", (T_REAL8)5.0); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedReal8Array

Name	TipcMsgAddNamedReal8Array — add a field containing an array of REAL8 fields to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedReal8Array(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_REAL8 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedReal8Array fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedReal8Array adds a field containing an array of REAL8 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendReal8Array, TipcMsgUpdateNamedReal8Array, TipcMsgGetNamedReal8Array

Examples This example creates a message and adds a REAL8_ARRAY value to it:

```
T_IPC_MSG msg;
T_BOOL status;
T_REAL8 *array;
T_INT4 array_size;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

array_size = 2;
array[0] = 56.1;
array[1] = 57.2;

status = TipcMsgAddNamedReal8Array(msg, "foo", array, array_size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedReal8ArrayPtr

Name	TipcMsgAddNamedReal8ArrayPtr — add a field containing an array of REAL8 fields to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedReal8ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_REAL8 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedReal8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedReal8ArrayPtr adds a field containing an array of REAL8 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendReal8ArrayPtr, TipcMsgUpdateNamedReal8ArrayPtr, TipcMsgFieldUpdateReal8ArrayPtr

Examples This example creates a message and adds a REAL8_ARRAY value to it:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL8 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_REAL8_ARRAY_TEST 100
mt = TipcMtCreate("real8_array_test", USER_MT_REAL8_ARRAY_TEST,
                 "real8_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedReal8ArrayPtr(msg, "named_real8_array", data,
0,
                                &field)) {
    return; /* error */
}

data[0] = 56.1;
data[1] = 57.2;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}
```


TipcMsgAddNamedReal16

Name	TipcMsgAddNamedReal16 — add a REAL16 field to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedReal16(msg, name, value) T_IPC_MSG msg; T_STR name; T_REAL16 value;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedReal16 fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedReal16 adds a REAL16 field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.



Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.

See Also TipcMsgAppendReal16, TipcMsgUpdateNamedReal16, TipcMsgGetNamedReal16

Examples This example creates a message and adds a REAL16 value to it:

```
T_IPC_MSG msg;
T_BOOL status;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

status = TipcMsgAddNamedReal16(msg, "foo", (T_REAL16)5.0);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedReal16Array

Name TipcMsgAddNamedReal16Array — add a field containing an array of REAL16 fields to a message using a name

Synopsis

```
T_BOOL TipcMsgAddNamedReal16Array(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_REAL16 *value;
T_INT4 size;
```

Arguments *msg* — message to add field to
name — name of the field
value — value of the field
size — number of elements in the field

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedReal16Array fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedReal16Array adds a field containing an array of REAL16 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.



Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.

See Also TipcMsgAppendReal16Array, TipcMsgUpdateNamedReal16Array, TipcMsgGetNamedReal16Array

Examples This example creates a message and adds a REAL16_ARRAY value to it:

```
T_IPC_MSG msg;
T_BOOL status;
T_REAL16 *array;
T_INT4 array_size;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

array_size = 2;
array[0] = 56.1;
array[1] = 57.2;

status = TipcMsgAddNamedReal16Array(msg, "foo", array,
array_size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedReal16ArrayPtr

Name TipcMsgAddNamedReal16ArrayPtr — add a field containing an array of REAL16 fields to a message using a name, and a pointer to the field value, rather than a copy

Synopsis T_BOOL TipcMsgAddNamedReal16ArrayPtr(*msg*, *name*, *value*, *size*, *field_return*)
 T_IPC_MSG *msg*;
 T_STR *name*;
 T_REAL16 **value*;
 T_INT4 *size*;
 T_IPC_MSG_FIELD **field_return*;

Arguments *msg* — message to add field to
 name — name of the field
 value — value of the field
 size — number of elements in the field
 field_return — optional value to return field (use NULL if not needed)

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedReal16ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedReal16ArrayPtr adds a field containing an array of REAL16 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.



Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.

See Also TipcMsgAppendReal16ArrayPtr, TipcMsgUpdateNamedReal16ArrayPtr, TipcMsgFieldUpdateReal16ArrayPtr

Examples This example creates a message and adds a REAL16_ARRAY value to it:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL16 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_REAL16_ARRAY_TEST 100
mt = TipcMtCreate("real16_array_test", USER_MT_REAL16_ARRAY_TEST,
                 "real16_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedReal16ArrayPtr(msg, "named_real16_array",
data, 0,
                                     &field)) {
    return; /* error */
}

data[0] = 1e1000;
data[1] = 2e1000;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}
```

TipcMsgAddNamedStr

Name TipcMsgAddNamedStr — add a STR field to a message using a name

Synopsis `T_BOOL TipcMsgAddNamedStr(msg, name, value)`
`T_IPC_MSG msg;`
`T_STR name;`
`T_STR value;`

Arguments *msg* — message to add field to
name — name of the field
value — value of the field

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedStr fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedStr adds a string field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.

See Also TipcMsgAppendStr, TipcMsgUpdateNamedStr, TipcMsgGetNamedStr

Examples This example creates a message and adds a STR value to it:

```
T_IPC_MSG msg;
T_BOOL status;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

status = TipcMsgAddNamedStr(msg, "foo", "bar");
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedStrArray

Name	TipcMsgAddNamedStrArray — add a field containing an array of STR fields to a message using a name
Synopsis	<pre> T_BOOL TipcMsgAddNamedStrArray(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_STR *value; T_INT4 size; </pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedStrArray fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedStrArray adds a field containing an array of STR fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendStrArray, TipcMsgUpdateNamedStrArray, TipcMsgGetNamedStrArray

Examples This example creates a message and adds a STR_ARRAY value to it:

```
T_IPC_MSG msg;
T_BOOL status;
T_STR *array;
T_INT4 array_size;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

array_size = 2;
array[0] = "Hello ";
array[1] = "World";

status = TipcMsgAddNamedStrArray(msg, "foo", array, array_size);
if (status == FALSE) {
    return; /* error */
}
```


TipcMsgAddNamedStrArrayPtr

Name	TipcMsgAddNamedStrArrayPtr — add a field containing an array of STR fields to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedStrArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_STR *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedStrArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedStrArrayPtr adds a field containing an array of STR fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendStrArrayPtr, TipcMsgUpdateNamedStrArrayPtr, TipcMsgFieldUpdateStrArrayPtr

Examples This example creates a message and adds a STR_ARRAY value to it:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR data[2];
T_IPC_MSG_FIELD field;

#define USER_MT_STR_ARRAY_TEST 100
mt = TipcMtCreate("str_array_test", USER_MT_STR_ARRAY_TEST,
                 "str_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedStrArrayPtr(msg, "named_str_array", data, 0,
                                &field)) {
    return; /* error */
}

data[0] = "Hello ";
data[1] = "world";
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}
```

TipcMsgAddNamedStrPtr

Name	TipcMsgAddNamedStrPtr — add a string field to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedStrPtr(msg, name, ptr, field_return) T_IPC_MSG msg; T_STR name; T_PTR ptr; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>ptr</i> — pointer to the value of the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedStrPtr fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedStrPtr adds a string field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendStrPtr, TipcMsgUpdateNamedStrPtr, TipcMsgFieldUpdateStrPtr
Examples	This example creates a message and adds a string value to it: <pre>T_IPC_MSG msg; T_BOOL status; T_IPC_MSG_FIELD *field_return; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } status = TipcMsgAddNamedStrPtr(msg, "foo", "bar", field_return); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedTimestamp

Name	TipcMsgAddNamedTimestamp — add a TIMESTAMP field to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedTimestamp(msg, name, value) T_IPC_MSG msg; T_STR name; T_REAL8 value;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedTimestamp fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedTimestamp adds a TIMESTAMP field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendTimestamp, TipcMsgUpdateNamedTimestamp, TipcMsgGetNamedTimestamp
Examples	This example creates a message and adds a TIMESTAMP value to it: <pre>T_IPC_MSG msg; T_BOOL status; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } status = TipcMsgAddNamedTimestamp(msg, "foo", TutGetWallTime()); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedTimestampArray

Name	TipcMsgAddNamedTimestampArray — add a field containing an array of TIMESTAMP fields to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedTimestampArray(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_REAL8 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedTimestampArray fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedTimestampArray adds a field containing an array of TIMESTAMP fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendTimestampArray, TipcMsgUpdateNamedTimestampArray, TipcMsgGetNamedTimestampArray

Examples This example creates a message and adds a `TIMESTAMP_ARRAY` value to it:

```
T_IPC_MSG msg;
T_BOOL status;
T_REAL8 *array;
T_INT4 array_size;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

array_size = 2;
array[0] = TutGetWallTime();
array[1] = TutGetWallTime() + 30;

status = TipcMsgAddNamedTimestampArray(msg, "foo", array,
array_size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedTimestampArrayPtr

Name	TipcMsgAddNamedTimestampArrayPtr — add a field containing an array of TIMESTAMP fields to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedTimestampArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_REAL8 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedTimestampArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedTimestampArrayPtr adds a field containing an array of TIMESTAMP fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendTimestampArrayPtr, TipcMsgUpdateNamedTimestampArrayPtr

Examples This example creates a message and adds a `TIMESTAMP_ARRAY` value to it:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL8 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_TIMESTAMP_ARRAY_TEST 100
mt = TipcMtCreate("timestamp_array_test",
USER_MT_TIMESTAMP_ARRAY_TEST,
"real8_array");

if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedTimestampArrayPtr(msg,
"named_timestamp_array",
                                data, 0, &field)) {
    return; /* error */
}

data[0] = TutGetWallTime();
data[1] = TutGetWallTime() + 30;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}
```


TipcMsgAddNamedUnknown

Name	TipcMsgAddNamedUnknown — add a field with an unknown or non-existent value to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedUnknown(msg, name, type) T_IPC_MSG msg; T_STR name; T_IPC_FT type;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>type</i> — type of the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedUnknown fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedUnknown adds a field with an unknown or non-existent value to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendUnknown, TipcMsgUpdateNamedUnknown, TipcMsgGetNamedUnknown
Examples	This example creates a message and adds an unknown value to it: <pre>T_IPC_MSG msg; T_BOOL status; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } status = TipcMsgAddNamedUnknown(msg, "foo", T_IPC_FT_REAL4); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedUtf8

Name TipcMsgAddNamedUtf8 — add a UTF8 field to a message using a name

Synopsis `T_BOOL TipcMsgAddNamedUtf8(msg, name, value)`
`T_IPC_MSG msg;`
`T_STR name;`
`T_STR value;`

Arguments *msg* — message to add field to
name — name of the field
value — value of the field

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedUtf8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedUtf8 adds a UTF8 field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.

See Also TipcMsgAppendUtf8, TipcMsgUpdateNamedUtf8, TipcMsgGetNamedUtf8

Examples This example creates a message and adds a UTF8 value to it:

```
T_IPC_MSG msg;
T_BOOL status;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

status = TipcMsgAddNamedUtf8(msg, "foo", "bar");
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedUtf8Array

Name	TipcMsgAddNamedUtf8Array — add a field containing an array of UTF8 fields to a message using a name
Synopsis	<pre>T_BOOL TipcMsgAddNamedUtf8Array(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_STR *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedUtf8Array fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedUtf8Array adds a field containing an array of UTF8 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.
See Also	TipcMsgAppendUtf8Array, TipcMsgUpdateNamedUtf8Array, TipcMsgGetNamedUtf8Array

Examples This example creates a message and adds a UTF8_ARRAY value to it:

```
T_IPC_MSG msg;
T_BOOL status;
T_STR *array;
T_INT4 array_size;

msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}

array_size = 2;
array[0] = "Hello ";
array[1] = "World";

status = TipcMsgAddNamedUtf8Array(msg, "foo", array, array_size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgAddNamedUtf8ArrayPtr

Name	TipcMsgAddNamedUtf8ArrayPtr — add a field containing an array of UTF8 fields to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedUtf8ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_STR *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedUtf8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedUtf8ArrayPtr adds a field containing an array of UTF8 fields to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendUtf8ArrayPtr, TipcMsgUpdateNamedUtf8ArrayPtr, TipcMsgFieldUpdateUtf8ArrayPtr

Examples This example creates a message and adds a UTF8_ARRAY value to it:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_UTF8_ARRAY_TEST 100
mt = TipcMtCreate("utf8_array_test", USER_MT_UTF8_ARRAY_TEST,
                 "utf8_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAddNamedUtf8ArrayPtr(msg, "named_utf8_array", data, 0,
                                &field)) {
    return; /* error */
}

data[0] = "Hello ";
data[1] = "World";
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}
```

TipcMsgAddNamedUtf8Ptr

Name	TipcMsgAddNamedUtf8Ptr — add a UTF8 field to a message using a name, and a pointer to the field value, rather than a copy
Synopsis	<pre>T_BOOL TipcMsgAddNamedUtf8Ptr(msg, name, ptr, field_return) T_IPC_MSG msg; T_STR name; T_PTR ptr; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to add field to <i>name</i> — name of the field <i>ptr</i> — pointer to the value of the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedUtf8Ptr fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field already exists in the message
Description	TipcMsgAddNamedUtf8Ptr adds a UTF8 field to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendUtf8Ptr, TipcMsgUpdateNamedUtf8Ptr, TipcMsgFieldUpdateUtf8Ptr
Examples	This example creates a message and adds a UTF8 value to it: <pre>T_IPC_MSG msg; T_BOOL status; T_IPC_MSG_FIELD *field_return; msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO)); if (msg == NULL) { return; /* error */ } status = TipcMsgAddNamedUtf8Ptr(msg, "foo", "bar", field_return); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgAddNamedXml

Name TipcMsgAddNamedXml — Add a named XML field to a message

Synopsis `T_BOOL TipcMsgAddNamedXml(msg, name, value)`
`T_IPC_MSG msg;`
`T_STR name;`
`T_XML value;`

Arguments *msg* — message to which to add the field
name — name of the field
value — value of the field

Return Values TRUE if the addition was successful, FALSE otherwise.

Diagnostics If TipcMsgAddNamedXml fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *msg*, *name* or *value* is null
- T_ERR_ALREADY_EXISTS — the field already exists in the message

Description TipcMsgAddNamedXml adds an XML object to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message.

See Also TipcMsgAppendXml, TipcMsgUpdateNamedXml, TipcMsgGetNamedXml

Examples

```
T_IPC_MSG msg;
T_XML xml;
T_BOOL status;
msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
if (msg == NULL) {
    return; /* error */
}
xml = TutXmlCreate("<data><data1><temp=\"hot\"/></data1></data>");
if (xml == NULL) {
    return; /* error */
}
status = TipcMsgAddNamedXml(msg, "foo", xml);
if (status == FALSE) {
    return; /* error */
}
```


TipcMsgAddNamedXmlPtr

Name	TipcMsgAddNamedXmlPtr — Add a named XML pointer field to a message
Synopsis	<pre>T_BOOL TipcMsgAddNamedXmlPtr(msg, name, value, field_return) T_IPC_MSG msg; T_STR name; T_XML value; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to which to add the field <i>name</i> — name of the field <i>value</i> — XML object to add to message <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the addition was successful, FALSE otherwise.
Diagnostics	If TipcMsgAddNamedXmlPtr fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_ALREADY_EXISTS — the field already exists in the message • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i>, or <i>value</i> is null
Description	TipcMsgAddNamedXmlPtr adds an XML object to a message, associating that field with a name. If the name already exists in the message, this fails, and the field is not added to the message. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAppendXmlPtr, TipcMsgUpdateNamedXmlPtr, TipcMsgFieldUpdateXmlPtr

```

Examples      T_IPC_MSG msg;
                  T_XML xml;
                  T_BOOL status;
                  T_IPC_MSG_FIELD *field_return;
                  msg = TipcMsgCreate(TipcMtLookupByNum(T_MT_INFO));
                  if (msg == NULL) {
                      return; /* error */
                  }
                  xml = TutXmlCreate("<data><data1><temp=\"hot\"/></data1></data>");
                  if (xml == NULL) {
                      return; /* error */
                  }
                  status = TipcMsgAddNamedXmlPtr(msg, "foo", xml, field_return);
                  if (status == FALSE) {
                      return; /* error */
                  }

```

TipcMsgAppend*

This chapter covers these TipcMsg API functions:

- *TipcMsgAppendBinary*
- *TipcMsgAppendBinaryPtr*
- *TipcMsgAppendBool*
- *TipcMsgAppendBoolArray*
- *TipcMsgAppendBoolArrayPtr*
- *TipcMsgAppendByte*
- *TipcMsgAppendChar*
- *TipcMsgAppendInt2*
- *TipcMsgAppendInt2Array*
- *TipcMsgAppendInt2ArrayPtr*
- *TipcMsgAppendInt4*
- *TipcMsgAppendInt4Array*
- *TipcMsgAppendInt4ArrayPtr*
- *TipcMsgAppendInt8*
- *TipcMsgAppendInt8Array*
- *TipcMsgAppendInt8ArrayPtr*
- *TipcMsgAppendMsg*
- *TipcMsgAppendMsgArray*
- *TipcMsgAppendMsgArrayPtr*
- *TipcMsgAppendMsgPtr*
- *TipcMsgAppendReal4*
- *TipcMsgAppendReal4Array*
- *TipcMsgAppendReal4ArrayPtr*
- *TipcMsgAppendReal8*
- *TipcMsgAppendReal8Array*

- *TipcMsgAppendReal8ArrayPtr*
- *TipcMsgAppendReal16*
- *TipcMsgAppendReal16Array*
- *TipcMsgAppendReal16ArrayPtr*
- *TipcMsgAppendStr*
- *TipcMsgAppendStrArray*
- *TipcMsgAppendStrArrayPtr*
- *TipcMsgAppendStrPtr*
- *TipcMsgAppendStrReal8*
- *TipcMsgAppendTimestamp*
- *TipcMsgAppendTimestampArray*
- *TipcMsgAppendTimestampArrayPtr*
- *TipcMsgAppendUnknown*
- *TipcMsgAppendUtf8*
- *TipcMsgAppendUtf8Array*
- *TipcMsgAppendUtf8ArrayPtr*
- *TipcMsgAppendUtf8Ptr*
- *TipcMsgAppendXml*
- *TipcMsgAppendXmlPtr*

TipcMsgAppendBinary

Name	TipcMsgAppendBinary — append a BINARY field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendBinary(msg, binary_data, binary_size) T_IPC_MSG msg; T_PTR binary_data; T_INT4 binary_size;</pre>
Arguments	<i>msg</i> — message to which to append the field <i>binary_data</i> — data for new message field <i>binary_size</i> — number of bytes in <i>binary_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendBinary fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>binary_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>binary_size</i> was negative
Description	TipcMsgAppendBinary appends a BINARY field containing <i>binary_data</i> to the end of a message's data. A binary field is a special case of an array of characters. Binary fields are useful for transferring images or large amounts of opaque data (such as an entire binary file). TipcMsgAppendBinary makes a copy of the binary data. Binary fields can be used for transferring actual C or C++ data structures, but the recommended approach is to use a separate message field for each data structure field. The separate fields allow SmartSockets to perform data conversion in a heterogeneous network environment. If the layout of the message must be determined dynamically, a message type grammar of "verbose" can be used. There is a small amount of overhead associated with each message field, so binary fields can occasionally be useful for large amounts of data (such as a large array of data structures containing several different-sized data types) sent between processes in a homogeneous network.
Caution	Because binary fields can contain any kind of data, there is no way that Tipc* functions or RTserver can byte-swap integers or convert floating-point numbers. It is up to the application to take care of this when sending binary fields.

See Also TipcMsgNextBinary

Examples This example creates a message and appends a BINARY image field:

```
T_IPC_MSG msg;
char image[1024][1024];

#define USER_MT_BINARY_TEST 100
mt = TipcMtCreate("binary_test", USER_MT_BINARY_TEST, "binary");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

fill_image(image); /* put data in the image */
if (!TipcMsgAppendBinary(msg, image, sizeof(image))) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendBinaryPtr

Name	TipcMsgAppendBinaryPtr — use a pointer to append a BINARY field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendBinaryPtr(msg, binary_data, binary_size, field_return) T_IPC_MSG msg; T_PTR binary_data; T_INT4 binary_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>binary_data</i> — data for new message field <i>binary_size</i> — number of bytes in <i>binary_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendBinaryPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>binary_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>binary_size</i> was negative
Description	TipcMsgAppendBinaryPtr appends a BINARY field containing <i>binary_data</i> to the end of a message's data. TipcMsgAppendBinaryPtr does not make a copy of the binary data. Instead, the supplied pointer to the data is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendBinaryPtr over TipcMsgAppendBinary is that the binary data can be modified after the field is appended, but before the message is sent. Another advantage is that the overhead of one copy operation is avoided. The data is still copied into the connection write buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateBinaryPtr function.

Caution Because binary fields can contain any kind of data, there is no way that Tipc* functions or RTserver can byte-swap integers or convert floating-point numbers. It is up to the application to take care of this when sending binary fields.

See Also TipcMsgAppendBinary, TipcMsgNextBinary

Examples This example creates a message and appends a BINARY image field:

```
T_IPC_MSG msg;
char image[1024][1024];
T_IPC_MSG_FIELD field;
T_INT4 image_size;

#define USER_MT_BINARY_TEST 100
mt = TipcMtCreate("binary_test", USER_MT_BINARY_TEST, "binary");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendBinaryPtr(msg, image, sizeof(image), &field)) {
    return; /* error */
}

image_size = fill_image(image); /* put data in the image */
if (!TipcMsgFieldSetSize(field, image_size)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```


TipcMsgAppendBool

Name	TipcMsgAppendBool — append a BOOL field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendBool(msg, bool_data) T_IPC_MSG msg; T_BOOL bool_data;</pre>
Arguments	<i>msg</i> — message to append field to <i>bool_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendBool fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendBool appends a BOOL field containing <i>bool_data</i> to the end of a message's data.
Caution	None
See Also	TipcMsgNextBool

Examples This example creates a message and appends a BOOL field:

```
T_IPC_MSG mt;
T_IPC_MSG msg;

#define USER_MT_BOOL_TEST 100
mt = TipcMtCreate("bool_test", USER_MT_BOOL_TEST,
                 "bool");

if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendBool(msg, T_TRUE)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendBoolArray

Name	TipcMsgAppendBoolArray — append a field containing an array of BOOL fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendBoolArray(msg, bool_array_data, array_size) T_IPC_MSG msg; T_BOOL *bool_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>bool_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>bool_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendBoolArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>bool_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendBoolArray appends a BOOL_ARRAY field containing <i>bool_array_data</i> to the end of a message's data. TipcMsgAppendBoolArray makes a copy of the BOOL_ARRAY data.
Caution	None
See Also	TipcMsgAppendBoolArrayPtr, TipcMsgNextBoolArray

Examples This example creates a message and appends a BOOL_ARRAY field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_BOOL data[2];

#define USER_MT_BOOL_ARRAY_TEST 100
mt = TipcMtCreate("bool_array_test", USER_MT_BOOL_ARRAY_TEST,
                 "bool_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data[0] = T_TRUE;
data[1] = T_FALSE;
if (!TipcMsgAppendboolArray(msg, data, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendBoolArrayPtr

Name	TipcMsgAppendBoolArrayPtr — use a pointer to append a field containing an array of BOOL fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendBoolArrayPtr(msg, bool_array_data, array_size, field_return) T_IPC_MSG msg; T_BOOL *bool_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>bool_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>bool_array_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendBoolArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>bool_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendBoolArrayPtr appends a BOOL_ARRAY field containing <i>bool_array_data</i> to the end of a message's data. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendBoolArrayPtr over TipcMsgAppendBoolArray is that the data may be modified after the field is appended, but before the message is sent. Also, the overhead of one copy operation is avoided, because the array is still copied into the connection message buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateBoolArrayPtr function.

Caution If the *field_return* argument is used to gain access to the message field, and the size of the field is subsequently modified using *TipcMsgFieldSetSize*, the *size* argument must be specified in elements rather than bytes.

See Also *TipcMsgAppendBoolArray*, *TipcMsgNextBoolArray*, *TipcMsgFieldSetSize*, *TipcMsgFieldUpdateBoolArrayPtr*

Examples This example creates a message and appends a pointer to a *BOOL_ARRAY* field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_BOOL data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_BOOL_ARRAY_TEST 100
mt = TipcMtCreate("bool_array_test", USER_MT_BOOL_ARRAY_TEST,
                 "bool_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendboolArrayPtr(msg, data, 0, &field)) {
    return; /* error */
}

data[0] = T_TRUE;
data[1] = T_FALSE;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendByte

Name	TipcMsgAppendByte — append a BYTE field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendByte(msg, byte_data) T_IPC_MSG msg; T_UCHAR byte_data;</pre>
Arguments	<i>msg</i> — message to which to append the field <i>byte_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendByte fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>byte_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendByte appends a BYTE field containing <i>byte_data</i> to the end of a message's data. TipcMsgAppendByte makes a copy of the byte data.
Caution	None
See Also	TipcMsgNextByte

Examples This example creates a message and appends a BYTE field:

```
T_IPC_MSG mt;
T_IPC_MSG msg;
T_UCHAR data;

#define USER_MT_BYTE_TEST 100
mt = TipcMtCreate("byte_test", USER_MT_BYTE_TEST,
                 "byte");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data=0xFF
if (!TipcMsgAppendByte(msg, data)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```


TipcMsgAppendChar

Name	TipcMsgAppendChar — append a CHAR field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendChar(msg, char_data) T_IPC_MSG msg; T_CHAR char_data;</pre>
Arguments	<i>msg</i> — message to append field to <i>char_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendChar fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendChar appends a CHAR field containing <i>char_data</i> to the end of a message's data.
Caution	None
See Also	TipcMsgNextChar

Examples This example creates a message and appends a CHAR field:

```
T_IPC_MT mt;
T_IPC_MSG msg;

#define USER_MT_CHAR_TEST 100
mt = TipcMtCreate("char_test", USER_MT_CHAR_TEST, "char");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendChar(msg, 'x')) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendInt2

Name	TipcMsgAppendInt2 — append an INT2 field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendInt2(msg, int2_data) T_IPC_MSG msg; T_INT2 int2_data;</pre>
Arguments	<i>msg</i> — message to append field to <i>int2_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendInt2 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendInt2 appends a two-byte integer (INT2) field containing <i>int2_data</i> to the end of a message's data.
Caution	None
See Also	TipcMsgNextInt2

Examples This example creates a message and appends an INT2 field:

```
T_IPC_MT mt;
T_IPC_MSG msg;

#define USER_MT_INT2_TEST 100
mt = TipcMtCreate("int2_test", USER_MT_INT2_TEST, "int2");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendInt2(msg, 42)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendInt2Array

Name	TipcMsgAppendInt2Array — append a field containing an array of INT2 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendInt2Array(msg, int2_array_data, array_size) T_IPC_MSG msg; T_INT2 *int2_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>int2_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>int2_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendInt2Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>int2_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendInt2Array appends a two-byte integer array (INT2_ARRAY) field containing <i>int2_array_data</i> to the end of a message's data. TipcMsgAppendInt2Array makes a copy of the array data.
Caution	None
See Also	TipcMsgAppendInt2ArrayPtr, TipcMsgNextInt2Array

Examples This example creates a message and appends an INT2_ARRAY field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT2 data[2];

#define USER_MT_INT2_ARRAY_TEST 100
mt = TipcMtCreate("int2_array_test", USER_MT_INT2_ARRAY_TEST,
                 "int2_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data[0] = 56;
data[1] = 57;
if (!TipcMsgAppendInt2Array(msg, data, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendInt2ArrayPtr

Name	TipcMsgAppendInt2ArrayPtr — use a pointer to append a field containing an array of INT2 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendInt2ArrayPtr(msg, int2_array_data, array_size, field_return) T_IPC_MSG msg; T_INT2 *int2_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>int2_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>int2_array_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendInt2ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>int2_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendInt2ArrayPtr appends a two-byte integer array (INT2_ARRAY) field containing <i>int2_array_data</i> to the end of a message's data. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendInt2ArrayPtr over TipcMsgAppendInt2Array is that the array data can be modified after the field is appended, but before the message is sent. Another advantage is that the overhead of one copy operation is avoided. The data is still copied into the connection write buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateInt2ArrayPtr function.

See Also `TipcMsgAppendInt2Array`, `TipcMsgNextInt2Array`, `TipcMsgFieldSetSize`, `TipcMsgFieldUpdateInt2ArrayPtr`

Examples This example creates a message and appends a pointer to an `INT2_ARRAY` field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT2 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_INT2_ARRAY_TEST 100
mt = TipcMtCreate("int2_array_test", USER_MT_INT2_ARRAY_TEST,
                 "int2_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendInt2ArrayPtr(msg, data, 0, &field)) {
    return; /* error */
}

data[0] = 56;
data[1] = 57;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```


TipcMsgAppendInt4

Name	TipcMsgAppendInt4 — append a field containing an array of INT4 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendInt4(msg, int4_data) T_IPC_MSG msg; T_INT4 int4_data;</pre>
Arguments	<i>msg</i> — message to append field to <i>int4_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendInt4 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendInt4 appends a four-byte integer (INT4) field containing <i>int4_data</i> to the end of a message's data.
Caution	None
See Also	TipcMsgNextInt4

Examples This example creates a message and appends an INT4 field:

```
T_IPC_MT mt;
T_IPC_MSG msg;

#define USER_MT_INT4_TEST 100
mt = TipcMtCreate("int4_test", USER_MT_INT4_TEST, "int4");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendInt4(msg, 42000000)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendInt4Array

Name	TipcMsgAppendInt4Array — append a field containing an array of INT4 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendInt4Array(msg, int4_array_data, array_size) T_IPC_MSG msg; T_INT4 *int4_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>int4_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>int4_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendInt4Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>int4_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendInt4Array appends a four-byte integer array (INT4_ARRAY) field containing <i>int4_array_data</i> to the end of a message's data. TipcMsgAppendInt4Array makes a copy of the array data.
Caution	None
See Also	TipcMsgAppendInt4ArrayPtr, TipcMsgNextInt4Array

Examples This example creates a message and appends an INT4_ARRAY field:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_INT4 data[2];

#define USER_MT_INT4_ARRAY_TEST 100
mt = TipcMtCreate("int4_array_test", USER_MT_INT4_ARRAY_TEST,
                 "int4_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data[0] = 560000;
data[1] = 570000;
if (!TipcMsgAppendInt4Array(msg, data, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```

TipcMsgAppendInt4ArrayPtr

Name	TipcMsgAppendInt4ArrayPtr — use a pointer to append a field containing an array of INT4 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendInt4ArrayPtr(msg, int4_array_data, array_size, field_return) T_IPC_MSG msg; T_INT4 *int4_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>int4_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>int4_array_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendInt4ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>int4_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendInt4ArrayPtr appends a four-byte integer array (INT4_ARRAY) field containing <i>int4_array_data</i> to the end of a message's data. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendInt4ArrayPtr over TipcMsgAppendInt4Array is that the array data can be modified after the field is appended, but before the message is sent. Also, the overhead of one copy operation is avoided, because the data is still copied into the connection write buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateInt4ArrayPtr function.

See Also `TipcMsgAppendInt4Array`, `TipcMsgNextInt4Array`, `TipcMsgFieldSetSize`, `TipcMsgFieldUpdateInt4ArrayPtr`

Examples This example creates a message and appends an `INT4_ARRAY` field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT4 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_INT4_ARRAY_TEST 100
mt = TipcMtCreate("int4_array_test", USER_MT_INT4_ARRAY_TEST,
                 "int4_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendInt4ArrayPtr(msg, data, 0, &field)) {
    return; /* error */
}

data[0] = 560000;
data[1] = 570000;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendInt8

Name	TipcMsgAppendInt8 — append an INT8 field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendInt8(msg, int8_data) T_IPC_MSG msg; T_INT8 int8_data;</pre>
Arguments	<i>msg</i> — message to append field to <i>int8_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendInt8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendInt8 appends an eight-byte integer (INT8) field containing <i>int8_data</i> to the end of a message's data.
Caution	Eight-byte integers are not available as native integer data types on all platforms.
See Also	TipcMsgNextInt8

Examples This example creates a message and appends an INT8 field:

```
T_IPC_MT mt;
T_IPC_MSG msg;

#define USER_MT_INT8_TEST 100
mt = TipcMtCreate("int8_test", USER_MT_INT8_TEST, "int8");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendInt8(msg, 42000000000)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```


TipcMsgAppendInt8Array

Name	TipcMsgAppendInt8Array — append an INT8_ARRAY field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendInt8Array(msg, int8_array_data, array_size) T_IPC_MSG msg; T_INT8 *int8_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>int8_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>int8_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendInt8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>int8_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendInt8Array appends an eight-byte integer array (INT8_ARRAY) field containing <i>int8_array_data</i> to the end of a message's data. TipcMsgAppendInt8Array makes a copy of the array data.
Caution	Eight-byte integers are not available as native integer data types on all platforms.
See Also	TipcMsgAppendInt8ArrayPtr, TipcMsgNextInt8Array

Examples This example creates a message and appends an INT8_ARRAY field:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_INT8 data[2];

#define USER_MT_INT8_ARRAY_TEST 100
mt = TipcMtCreate("int8_array_test", USER_MT_INT8_ARRAY_TEST,
                 "int8_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data[0] = 56000000000;
data[1] = 57000000000;
if (!TipcMsgAppendInt8Array(msg, data, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```

TipcMsgAppendInt8ArrayPtr

Name	TipcMsgAppendInt8ArrayPtr — use a pointer to append an INT8_ARRAY field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendInt8ArrayPtr(msg, int8_array_data, array_size, field_return) T_IPC_MSG msg; T_INT8 *int8_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>int8_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>int8_array_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendInt8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>int8_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendInt8ArrayPtr appends an eight-byte integer array (INT8_ARRAY) field containing <i>int8_array_data</i> to the end of a message's data. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendInt8ArrayPtr over TipcMsgAppendInt8Array is that the array data can be modified after the field is appended, but before the message is sent. Another advantage is that the overhead of one copy operation is avoided. The data is still copied into the connection write buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateInt8ArrayPtr function.

Caution Eight-byte integers are not available as native integer data types on all platforms.

See Also `TipcMsgAppendInt8Array`, `TipcMsgNextInt8Array`, `TipcMsgFieldSetSize`, `TipcMsgFieldUpdateInt8ArrayPtr`

Examples This example creates a message and appends an `INT8_ARRAY` field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT8 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_INT8_ARRAY_TEST 100
mt = TipcMtCreate("int8_array_test", USER_MT_INT8_ARRAY_TEST,
                 "int8_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendInt8ArrayPtr(msg, data, 0, &field)) {
    return; /* error */
}

data[0] = 56000000000;
data[1] = 57000000000;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendMsg

Name	TipcMsgAppendMsg — append a MSG field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendMsg(msg, msg_data) T_IPC_MSG msg; T_IPC_MSG msg_data;</pre>
Arguments	<i>msg</i> — message to append field to <i>msg_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendMsg fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>msg_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message or <i>msg_data</i> was not a valid message • T_ERR_VAL_INVALID — <i>msg_data</i> and <i>msg</i> were the same message (a message cannot be appended to itself)
Description	TipcMsgAppendMsg appends a MSG field containing <i>msg_data</i> to the end of a message's data. TipcMsgAppendMsg makes a copy of the message data.
Caution	None
See Also	TipcMsgAppendMsgPtr, TipcMsgNextMsg

Examples This example creates a message and appends a MSG field:

```
T_IPC_MSG msg;
T_IPC_MSG msg_data;

#define USER_MT_CONTAINER 100

mt = TipcMtCreate("container", USER_MT_CONTAINER, "msg");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}
msg_data = TipcMsgCreate(mt);
if (msg_data == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg_data, "speed_limit", 55.0)) {
    return; /* error */
}

if (!TipcMsgAppendMsg(msg, msg_data)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendMsgArray

Name	TipcMsgAppendMsgArray — append a field containing an array of MSG fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendMsgArray(msg, msg_array_data, array_size) T_IPC_MSG msg; T_IPC_MSG *msg_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>msg_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>msg_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendMsgArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>msg_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message or one of the messages in <i>msg_array_data</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative • T_ERR_VAL_INVALID — one of the messages in <i>msg_array_data</i> and <i>msg</i> were the same message (a message cannot be appended to itself)
Description	TipcMsgAppendMsgArray appends a message array (MSG_ARRAY) field containing <i>msg_array_data</i> to the end of a message's data. TipcMsgAppendMsgArray makes a copy of the message array data.
Caution	None
See Also	TipcMsgAppendMsgArrayPtr, TipcMsgNextMsgArray

Examples This example creates a message and appends a MSG_ARRAY field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
#define ARRAY_SIZE 3
T_IPC_MSG msg_array_data[ARRAY_SIZE];

#define USER_MT_CONTAINER 3
mt = TipcMtCreate("container", USER_MT_CONTAINER, "msg_array");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

mt = TipcMtLookupByNum(T_MT_TIME);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[0] = TipcMsgCreate(mt);
if (msg_array_data[0] == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[1] = TipcMsgCreate(mt);
if (msg_array_data[1] == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_END_OF_FRAME);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[2] = TipcMsgCreate(mt);
if (msg_array_data[2] == NULL) {
    return; /* error */
}

if (!TipcMsgAppendReal8(msg_array_data[0], TutGetWallTime())) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg_array_data[1], "speed_limit",
55.0)) {
    return; /* error */
}
```



```
if (!TipcMsgAppendMsgArray(msg, msg_array_data, ARRAY_SIZE)) {  
    return; /* error */  
}  
  
/* A process would typically send a message at this point by */  
/* calling TipcConnMsgSend or TipcSrvMsgSend. */  
  
/* A process would typically destroy a message at this point by */  
/* calling TipcMsgDestroy. */
```

TipcMsgAppendMsgArrayPtr

Name	TipcMsgAppendMsgArrayPtr — use a pointer to append a field containing an array of MSG fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendMsgArrayPtr(msg, msg_array_data, array_size, field_return) T_IPC_MSG msg; T_IPC_MSG *msg_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>msg_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>msg_array_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendMsgArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>msg_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message or one of the messages in <i>msg_array_data</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>msg_array_size</i> was negative • T_ERR_VAL_INVALID — one of the messages in <i>msg_array_data</i> and <i>msg</i> were the same message (a message cannot be appended to itself)
Description	TipcMsgAppendMsgArrayPtr appends a message array (MSG_ARRAY) field containing <i>msg_array_data</i> to the end of a message’s data. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field’s internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.

One advantage of `TipcMsgAppendMsgArrayPtr` over `TipcMsgAppendMsgArray` is that the array messages can be modified after the field is appended, but before the message is sent. Also, the overhead of one copy operation is avoided, because the array is still copied into the connection write buffer when the message is sent.

To update the pointer to point to a new memory location, use the `TipcMsgFieldUpdateMsgArrayPtr` function.

Caution If the *field_return* argument is used to gain access to the message field, and the size of the field is subsequently modified using `TipcMsgFieldSetSize`, the *array_size* argument must be specified in elements rather than bytes.

See Also `TipcMsgAppendMsgArray`, `TipcMsgNextMsgArray`, `TipcMsgFieldSetSize`, `TipcMsgFieldUpdateMsgArrayPtr`

Examples This example creates a message and appends a pointer to a `MSG_ARRAY` field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
#define ARRAY_SIZE 3
T_IPC_MSG msg_array_data[ARRAY_SIZE];

#define USER_MT_CONTAINER 3
mt = TipcMtCreate("container", USER_MT_CONTAINER, "msg_array");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

mt = TipcMtLookupByNum(T_MT_TIME);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[0] = TipcMsgCreate(mt);
if (msg_array_data[0] == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[1] = TipcMsgCreate(mt);
if (msg_array_data[1] == NULL) {
    return; /* error */
}
```

```

mt = TipcMtLookupByNum(T_MT_END_OF_FRAME);
if (mt == NULL) {
    return; /* error */
}
msg_array_data[2] = TipcMsgCreate(mt);
if (msg_array_data[2] == NULL) {
    return; /* error */
}

if (!TipcMsgAppendMsgArrayPtr(msg, msg_array_data, ARRAY_SIZE,
                             NULL)) {
    return; /* error */
}

if (!TipcMsgAppendReal8(msg_array_data[0], TutGetWallTime())) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg_array_data[1], "speed_limit",
70.0)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```

TipcMsgAppendMsgPtr

Name	TipcMsgAppendMsgPtr — use a pointer to append a MSG field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendMsgPtr(msg, msg_data, field_return) T_IPC_MSG msg; T_IPC_MSG msg_data; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>msg_data</i> — data for new message field <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendMsgPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>msg_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message or <i>msg_data</i> was not a valid message • T_ERR_VAL_INVALID — <i>msg_data</i> and <i>msg</i> were the same message (a message cannot be appended to itself)
Description	TipcMsgAppendMsgPtr appends a MSG field containing <i>msg_data</i> to the end of a message's data. This function does not make a copy of the message. Instead, the supplied pointer is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for destroying the inner message once the field has been destroyed. One advantage of TipcMsgAppendMsgPtr over TipcMsgAppendMsg is that the inner message can be modified after the field is appended, but before the message is sent. Another advantage is that the overhead of one copy operation is avoided. The inner message is still copied into the connection write buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateMsgPtr function.
Caution	None

See Also `TipcMsgAppendMsg`, `TipcMsgNextMsg`, `TipcMsgFieldSetSize`,
`TipcMsgFieldUpdateMsgPtr`

Examples This example creates a message and appends a MSG field:

```
T_IPC_MSG msg;
T_IPC_MSG msg_data;

#define USER_MT_CONTAINER 100

mt = TipcMtCreate("container", USER_MT_CONTAINER, "msg");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}
msg_data = TipcMsgCreate(mt);
if (msg_data == NULL) {
    return; /* error */
}

if (!TipcMsgAppendMsgPtr(msg, msg_data, NULL)) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg_data, "speed_limit", 70.0)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendReal4

Name	TipcMsgAppendReal4 — append a REAL4 field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendReal4(<i>msg</i>, <i>real4_data</i>) T_IPC_MSG <i>msg</i>; T_REAL4 <i>real4_data</i>;</pre>
Arguments	<i>msg</i> — message to append field to <i>real4_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendReal4 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendReal4 appends a four-byte real number (REAL4) field containing <i>real4_data</i> to the end of a message's data.
Caution	None
See Also	TipcMsgNextReal4

Examples This example creates a message and appends a REAL4 field:

```
T_IPC_MT mt;
T_IPC_MSG msg;

#define USER_MT_REAL4_TEST 100
mt = TipcMtCreate("real4_test", USER_MT_REAL4_TEST, "real4");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}
/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendReal4(msg, 42.0)) {
    return; /* error */
}
/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */
/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```


TipcMsgAppendReal4Array

Name	TipcMsgAppendReal4Array — append a field containing an array of REAL4 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendReal4Array(msg, real4_array_data, array_size) T_IPC_MSG msg; T_REAL4 *real4_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>real4_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>real4_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendReal4Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>real4_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendReal4Array appends a four-byte real number array (REAL4_ARRAY) field containing <i>real4_array_data</i> to the end of a message's data. TipcMsgAppendReal4Array makes a copy of the array data.
Caution	None
See Also	TipcMsgAppendReal4ArrayPtr, TipcMsgNextReal4Array

Examples This example creates a message and appends a REAL4_ARRAY field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL4 data[2];

#define USER_MT_REAL4_ARRAY_TEST 100
mt = TipcMtCreate("real4_array_test", USER_MT_REAL4_ARRAY_TEST,
                 "real4_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data[0] = 56.1;
data[1] = 57.2;
if (!TipcMsgAppendReal4Array(msg, data, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendReal4ArrayPtr

Name	TipcMsgAppendReal4ArrayPtr — use a pointer to append a field containing an array of REAL4 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendReal4ArrayPtr(msg, real4_array_data, array_size, field_return) T_IPC_MSG msg; T_REAL4 *real4_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>real4_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>real4_array_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendReal4ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>real4_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendReal4ArrayPtr appends a four-byte real array (REAL4_ARRAY) field containing <i>real4_array_data</i> to the end of a message's data. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendReal4ArrayPtr over TipcMsgAppendReal4Array is that the array data can be modified after the field is appended, but before the message is sent. Also, the overhead of one copy operation is avoided, because the data is still copied into the connection write buffer when the message is sent.

To update the pointer to point to a new memory location, use the `TipcMsgFieldUpdateReal4ArrayPtr` function.

See Also `TipcMsgAppendReal4Array`, `TipcMsgNextReal4Array`, `TipcMsgFieldSetSize`, `TipcMsgFieldUpdateReal4ArrayPtr`

Examples This example creates a message and appends a `REAL4_ARRAY` field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL4 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_REAL4_ARRAY_TEST 100
mt = TipcMtCreate("real4_array_test", USER_MT_REAL4_ARRAY_TEST,
                 "real4_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendReal4ArrayPtr(msg, data, 0, &field)) {
    return; /* error */
}

data[0] = 56.1;
data[1] = 57.2;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendReal8

Name	TipcMsgAppendReal8 — append a REAL8 field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendReal8(<i>msg</i>, <i>real8_data</i>) T_IPC_MSG <i>msg</i>; T_REAL8 <i>real8_data</i>;</pre>
Arguments	<i>msg</i> — message to append field to <i>real8_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendReal8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendReal8 appends an eight-byte real number (REAL8) field containing <i>real8_data</i> to the end of a message's data.
Caution	None
See Also	TipcMsgNextReal8

Examples This example creates a message and appends a REAL8 field:

```
T_IPC_MSG mt;
T_IPC_MSG msg;

#define USER_MT_REAL8_TEST 100
mt = TipcMtCreate("real8_test", USER_MT_REAL8_TEST,
                 "real8");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendReal8(msg, 42.0)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendReal8Array

Name	TipcMsgAppendReal8Array — append a field containing an array of REAL8 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendReal8Array(msg, real8_array_data, array_size) T_IPC_MSG msg; T_REAL8 *real8_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>real8_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>real8_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendReal8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>real8_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendReal8Array appends an eight-byte real number array (REAL8_ARRAY) field containing <i>real8_array_data</i> to the end of a message's data. TipcMsgAppendReal8Array makes a copy of the array data.
Caution	None
See Also	TipcMsgAppendReal8Array, TipcMsgNextReal8Array

Examples This example creates a message and appends a REAL8_ARRAY field:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL8 data[2];

#define USER_MT_REAL8_ARRAY_TEST 100
mt = TipcMtCreate("real8_array_test", USER_MT_REAL8_ARRAY_TEST,
                 "real8_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data[0] = 56.1;
data[1] = 57.2;
if (!TipcMsgAppendReal8Array(msg, data, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```


TipcMsgAppendReal8ArrayPtr

Name	TipcMsgAppendReal8ArrayPtr — use a pointer to append a field containing an array of REAL8 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendReal8ArrayPtr(msg, real8_array_data, array_size, field_return) T_IPC_MSG msg; T_REAL8 *real8_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>real8_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>real8_array_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendReal8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>real8_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendReal8ArrayPtr appends an eight-byte real array (REAL8_ARRAY) field containing <i>real8_array_data</i> to the end of a message's data. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendReal8ArrayPtr over TipcMsgAppendReal8Array is that the array data can be modified after the field is appended, but before the message is sent. Another advantage is that the overhead of one copy operation is avoided. The data is still copied into the connection write buffer when the message is sent.

To update the pointer to point to a new memory location, use the `TipcMsgFieldUpdateReal8ArrayPtr` function.

See Also `TipcMsgAppendReal8Array`, `TipcMsgNextReal8Array`, `TipcMsgFieldSetSize`, `TipcMsgFieldUpdateReal8ArrayPtr`

Examples This example creates a message and appends a `REAL8_ARRAY` field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL8 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_REAL8_ARRAY_TEST 100
mt = TipcMtCreate("real8_array_test", USER_MT_REAL8_ARRAY_TEST,
                 "real8_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendReal8ArrayPtr(msg, data, 0, &field)) {
    return; /* error */
}

data[0] = 56.1;
data[1] = 57.2;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendReal16

Name	TipcMsgAppendReal16 — append a REAL16 field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendReal16(msg, real16_data) T_IPC_MSG msg; T_REAL16 real16_data;</pre>
Arguments	<i>msg</i> — message to append field to <i>real16_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendReal16 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendReal16 appends a 16-byte real number (REAL16) field containing <i>real16_data</i> to the end of a message's data.
	Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.
See Also	TipcMsgNextReal16

Examples This example creates a message and appends a REAL16 field:

```
T_IPC_MSG mt;
T_IPC_MSG msg;

#define USER_MT_REAL16_TEST 100
mt = TipcMtCreate("real16_test", USER_MT_REAL16_TEST,
                 "real16");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendReal16(msg, 1e1000)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendReal16Array

Name	TipcMsgAppendReal16Array — append a field containing an array of REAL16 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendReal16Array(msg, real16_array_data, array_size) T_IPC_MSG msg; T_REAL16 *real16_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>real16_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>real16_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendReal16Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>real16_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendReal16Array appends a 16-byte real number array (REAL16_ARRAY) field containing <i>real16_array_data</i> to the end of a message's data. TipcMsgAppendReal16Array makes a copy of the array data.
	Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.
See Also	TipcMsgAppendReal16ArrayPtr, TipcMsgNextReal16Array

Examples This example creates a message and appends a REAL16_ARRAY field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL16 data[2];

#define USER_MT_REAL16_ARRAY_TEST 100
mt = TipcMtCreate("real16_array_test", USER_MT_REAL16_ARRAY_TEST,
                 "real16_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data[0] = 1e1000;
data[1] = 2e1000;
if (!TipcMsgAppendReal16Array(msg, data, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendReal16ArrayPtr

Name	TipcMsgAppendReal16ArrayPtr — use a pointer to append a field containing an array of REAL16 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendReal16ArrayPtr(msg, real16_array_data, array_size, field_return) T_IPC_MSG msg; T_REAL16 *real16_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>real16_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>real16_array_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendReal16ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>real16_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendReal16ArrayPtr appends a 16-byte real array (REAL16_ARRAY) field containing <i>real16_array_data</i> to the end of a message's data. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendReal16ArrayPtr over TipcMsgAppendReal16Array is that the array data can be modified after the field is appended, but before the message is sent. Also, the overhead of one copy operation is avoided, because the data is still copied into the connection message buffer when the message is sent.

To update the pointer to point to a new memory location, use the `TpcMsgFieldUpdateReal16ArrayPtr` function.



Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.

See Also `TpcMsgAppendReal16Array`, `TpcMsgNextReal16Array`, `TpcMsgFieldSetSize`, `TpcMsgFieldUpdateReal16ArrayPtr`

Examples This example creates a message and appends a REAL16_ARRAY field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL16 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_REAL16_ARRAY_TEST 100
mt = TpcMtCreate("real16_array_test", USER_MT_REAL16_ARRAY_TEST,
               "real16_array");
if (mt == NULL) {
    return; /* error */
}

msg = TpcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TpcMsgSetDest. */

if (!TpcMsgAppendReal16ArrayPtr(msg, data, 0, &field)) {
    return; /* error */
}

data[0] = 1e1000;
data[1] = 2e1000;
if (!TpcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TpcConnMsgSend or TpcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TpcMsgDestroy. */
```


TipcMsgAppendStr

Name	TipcMsgAppendStr — append a STR field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendStr(msg, str_data) T_IPC_MSG msg; T_STR str_data;</pre>
Arguments	<i>msg</i> — message to append field to <i>str_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendStr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>str_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendStr appends a null-terminated character string field containing <i>str_data</i> to the end of a message's data. A string field is a special case of an array of characters. TipcMsgAppendStr makes a copy of the string data.
Caution	None
See Also	TipcMsgAppendStrPtr, TipcMsgNextStr

Examples This example creates a message and appends a STR field:

```
T_IPC_MT mt;
T_IPC_MSG msg;

#define USER_MT_STR_TEST 100
mt = TipcMtCreate("str_test", USER_MT_STR_TEST, "str");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (mt == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendStr(msg, "This is the field data. ")) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendStrArray

Name	TipcMsgAppendStrArray — append a field containing an array of STR fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendStrArray(msg, str_array_data, array_size) T_IPC_MSG msg; T_STR *str_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>str_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>str_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendStrArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>str_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendStrArray appends a string array (STR_ARRAY) field containing <i>str_array_data</i> to the end of a message's data. TipcMsgAppendStrArray makes a copy of the string array data.
Caution	None
See Also	TipcMsgAppendStrArrayPtr, TipcMsgNextStrArray

Examples This example creates a message and appends a STR_ARRAY field:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_STR data[2];

#define USER_MT_STR_ARRAY_TEST 100
mt = TipcMtCreate("str_array_test", USER_MT_STR_ARRAY_TEST,
                 "str_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data[0] = "Hello";
data[1] = "world";
if (!TipcMsgAppendStrArray(msg, data, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```

TipcMsgAppendStrArrayPtr

Name	TipcMsgAppendStrArrayPtr — use a pointer to append a field containing an array of STR fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendStrArrayPtr(msg, str_array_data, array_size, field_return) T_IPC_MSG msg; T_STR *str_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>str_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>str_array_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendStrArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>str_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendStrArrayPtr appends a string array (STR_ARRAY) field containing <i>str_array_data</i> to the end of a message's data. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendStrArrayPtr over TipcMsgAppendStrArray is that the character strings can be modified after the field is appended, but before the message is sent. Also, the overhead of one copy operation is avoided, because the array is still copied into the connection message buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateStrArrayPtr function.

Caution If the *field_return* argument is used to gain access to the message field, and the size of the field is subsequently modified using *TipcMsgFieldSetSize*, the *size* argument must be specified in elements rather than bytes.

See Also *TipcMsgAppendStrArray*, *TipcMsgNextStrArray*, *TipcMsgFieldSetSize*, *TipcMsgFieldUpdateStrArrayPtr*

Examples This example creates a message and appends a STR_ARRAY field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR data[2];
T_IPC_MSG_FIELD field;

#define USER_MT_STR_ARRAY_TEST 100
mt = TipcMtCreate("str_array_test", USER_MT_STR_ARRAY_TEST,
                 "str_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendStrArrayPtr(msg, data, 0, &field)) {
    return; /* error */
}

data[0] = "Hello ";
data[1] = "world";
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendStrPtr

Name	TipcMsgAppendStrPtr — use a pointer to append a STR field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendStrPtr(msg, str_data, field_return) T_IPC_MSG msg; T_STR str_data; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>str_data</i> — data for new message field <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendStrPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>str_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendStrPtr appends a null-terminated character STR field containing <i>str_data</i> to the end of a message's data. This function does not make a copy of the string. Instead, the supplied pointer to the string is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendStrPtr over TipcMsgAppendStr is that the string can be modified after the field is appended, but before the message is sent. Also, the overhead of one copy operation is avoided, because the character string is still copied into the connection message buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateStrPtr function.
Caution	None
See Also	TipcMsgAppendStr, TipcMsgNextStr, TipcMsgFieldSetSize, TipcMsgFieldUpdateStrPtr

Examples This example creates a message and appends a pointer to a STR field:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_CHAR str[80];
T_IPC_MSG_FIELD field;

#define USER_MT_STR_TEST 100
mt = TipcMtCreate("str_test", USER_MT_STR_TEST, "str");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (mt == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

str[0] = '\0';
if (!TipcMsgAppendStrPtr(msg, str, &field)) {
    return; /* error */
}

strcpy(str, "This is the field data.");
if (!TipcMsgFieldSetSize(field, strlen(str) + 1)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```


TipcMsgAppendStrReal8

Name	TipcMsgAppendStrReal8 — append a STR field and a REAL8 field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendStrReal8(msg, str_data, real8_data) T_IPC_MSG msg; T_STR str_data; T_REAL8 real8_data;</pre>
Arguments	<i>msg</i> — message to append fields to <i>str_data</i> — data for new message field <i>real8_data</i> — data for new message field
Return Values	TRUE if the fields were successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendStrReal8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>str_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendStrReal8 is a convenience function that appends both a STR field and a REAL8 field to the end of a message's data. TipcMsgAppendStrReal8 is useful for constructing NUMERIC_DATA messages. The NUMERIC_DATA message type is often the most frequently used message type in SmartSockets. Using TipcMsgAppendStrReal8 is slightly faster than using TipcMsgAppendStr and TipcMsgAppendReal8.
Caution	None
See Also	TipcMsgNextStrReal8

Examples This example creates a message and appends STR and REAL8 fields:

```

T_IPC_MT mt;
T_IPC_MSG msg;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendStrReal8(msg, "voltage", 120.0)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```

TpcMsgAppendTimestamp

Name	TpcMsgAppendTimestamp — append a <code>TIMESTAMP</code> field to a message
Synopsis	<pre>T_BOOL TpcMsgAppendTimestamp(msg, timestamp_data) T_IPC_MSG msg; T_REAL8 timestamp_data;</pre>
Arguments	<i>msg</i> — message to append field to <i>timestamp_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TpcMsgAppendTimestamp fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_NULL_PTR</code> — <i>msg</i> was null • <code>T_ERR_READ_ONLY</code> — the message was marked read-only. This happens if the message is a field in another message. • <code>T_ERR_MSG_INVALID</code> — <i>msg</i> was not a valid message
Description	TpcMsgAppendTimestamp appends a timestamp, or the current computer wall clock time in the number of seconds since midnight January 1, 1970. The <code>TIMESTAMP</code> field is an eight-byte real number containing <i>timestamp_data</i> to the end of a message's data.
Caution	None
See Also	TpcMsgNextTimestamp

Examples This example creates a message and appends a TIMESTAMP field:

```
T_IPC_MSG mt;
T_IPC_MSG msg;

#define USER_MT_TIMESTAMP_TEST 100
mt = TipcMtCreate("timestamp_test", USER_MT_TIMESTAMP_TEST,
                 "timestamp");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendTimestamp(msg, TutGetWallTime())) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendTimestampArray

Name	TipcMsgAppendTimestampArray — append a field containing an array of <code>TIMESTAMP</code> fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendTimestampArray(msg, timestamp_array_data, array_size) T_IPC_MSG msg; T_REAL8 *timestamp_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>timestamp_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>timestamp_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendTimestampArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_NULL_PTR</code> — <i>msg</i> was null or <i>timestamp_array_data</i> was null • <code>T_ERR_READ_ONLY</code> — the message was marked read-only. This happens if the message is a field in another message. • <code>T_ERR_MSG_INVALID</code> — <i>msg</i> was not a valid message • <code>T_ERR_VAL_TOO_SMALL</code> — <i>array_size</i> was negative
Description	TipcMsgAppendTimestampArray appends a <code>TIMESTAMP_ARRAY</code> (an eight-byte real number array) field containing <i>timestamp_array_data</i> to the end of a message's data. A timestamp is the current computer wall clock time in the number of seconds since midnight January 1, 1970. TipcMsgAppendTimestampArray makes a copy of the array data.
Caution	None
See Also	TipcMsgAddNamedTimestampArray, TipcMsgNextTimestampArray

Examples This example creates a message and appends a `TIMESTAMP_ARRAY` field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL8 data[2];

#define USER_MT_TIMESTAMP_ARRAY_TEST 100
mt = TipcMtCreate("timestamp_array_test",
USER_MT_TIMESTAMP_ARRAY_TEST,
"timestamp_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data[0] = TutGetWallTime();
data[1] = TutGetWallTime() + 30;
if (!TipcMsgAppendTimestampArray(msg, data, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendTimestampArrayPtr

Name	TipcMsgAppendTimestampArrayPtr — use a pointer to append a field containing an array of <code>TIMESTAMP</code> fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendTimestampArrayPtr(msg, timestamp_array_data, array_size, field_return) T_IPC_MSG msg; T_REAL8 *timestamp_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>timestamp_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>timestamp_array_data</i> <i>field_return</i> — optional location to return field (use <code>NULL</code> if not needed)
Return Values	<code>TRUE</code> if the field was successfully appended to the message, <code>FALSE</code> otherwise.
Diagnostics	If <code>TipcMsgAppendTimestampArrayPtr</code> fails, it returns <code>FALSE</code> and sets the global SmartSockets error number to one of: <code>T_ERR_NULL_PTR</code> — <i>msg</i> was null or <i>timestamp_array_data</i> was null <code>T_ERR_READ_ONLY</code> — the message was marked read-only. This happens if the message is a field in another message. <code>T_ERR_MSG_INVALID</code> — <i>msg</i> was not a valid message <code>T_ERR_VAL_TOO_SMALL</code> — <i>array_size</i> was negative
Description	<code>TipcMsgAppendTimestampArrayPtr</code> appends a <code>TIMESTAMP_ARRAY</code> (an eight-byte real number array) field containing <i>timestamp_array_data</i> to the end of a message's data. A timestamp is the current computer wall clock time in the number of seconds since midnight January 1, 1970. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.

One advantage of `TipcMsgAppendTimestampArrayPtr` over `TipcMsgAppendTimestampArray` is that the array data can be modified after the field is appended, but before the message is sent. Another advantage is that the overhead of one copy operation is avoided. The data is still copied into the connection write buffer when the message is sent.

To update the pointer to point to a new memory location, use the `TipcMsgFieldUpdateTimestampArrayPtr` function.

See Also `TipcMsgAppendTimestampArray`, `TipcMsgNextTimestampArray`, `TipcMsgFieldSetSize`, `TipcMsgUpdateNamedTimestampArrayPtr`

Examples This example creates a message and appends a `TIMESTAMP_ARRAY` field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL8 data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_TIMESTAMP_ARRAY_TEST 100
mt = TipcMtCreate("timestamp_array_test",
USER_MT_TIMESTAMP_ARRAY_TEST,
"timestamp_array");

if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendTimestampArrayPtr(msg, data, 0, &field)) {
    return; /* error */
}

data[0] = TutGetWallTime();
data[1] = TutGetWallTime() + 30;
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```


TipcMsgAppendUnknown

Name	TipcMsgAppendUnknown — append a field to a message and set that field's value to unknown
Synopsis	<pre>T_BOOL TipcMsgAppendUnknown(msg, field_type) T_IPC_MSG msg; T_IPC_FT field_type;</pre>
Arguments	<i>msg</i> — message to append unknown-valued field to <i>field_type</i> — field type for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendUnknown fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>field_type</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendUnknown appends a field of the type specified to the end of a message's data, and sets the <i>unknown_value</i> property of the field to TRUE. Frequently, messages have a fixed number of fields in a constant order. In that case, omitting a field when data occasionally does not exist is not feasible. Setting the <i>unknown_value</i> property allows fields to be marked as having no data. This feature is most useful when the entire range of field values is valid within an application and there is a need to denote that the field has no valid data.
Caution	Only valid field types as defined in the enumerated structure T_IPC_FT are allowable values for the <i>field_type</i> argument. Property components of that structure indicated with the T_IPC_PROP prefix, such as destination (T_IPC_PROP_DEST) or sender (T_IPC_PROP_SENDER) cannot be appended as unknown values. See Messages on page 10 for a definition of this structure.
See Also	TipcMsgNextUnknown, TipcMsgGetCurrentFieldKnown

Examples This example creates a message and appends a field of unknown value:

```

T_IPC_MT mt;
T_IPC_MSG msg;

mt=TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);

if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendUnknown(msg, T_IPC_FT_INT4)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```

TipcMsgAppendUtf8

Name	TipcMsgAppendUtf8 — append a UTF8 field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendUtf8(msg, utf8_data) T_IPC_MSG msg; T_STR utf8_data;</pre>
Arguments	<i>msg</i> — message to append field to <i>utf8_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendStr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>str_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendStr appends a UTF8 field containing <i>utf8_data</i> to the end of a message's data. TipcMsgAppendUtf8 makes a copy of the UTF8 data.
Caution	None
See Also	TipcMsgAppendUtf8Ptr, TipcMsgNextUtf8

Examples This example creates a message and appends a UTF8 field:

```
T_IPC_MSG mt;
T_IPC_MSG msg;

#define USER_MT_UTF8_TEST 100
mt = TipcMtCreate("utf8_test", USER_MT_UTF8_TEST,
                 "utf8");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendUtf8(msg, "test")) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendUtf8Array

Name	TipcMsgAppendUtf8Array — append a field containing an array of UTF8 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendUtf8Array(msg, utf8_array_data, array_size) T_IPC_MSG msg; T_STR *utf8_array_data; T_INT4 array_size;</pre>
Arguments	<i>msg</i> — message to append field to <i>utf8_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>utf8_array_data</i>
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendUtf8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>utf8_array_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendutf8Array appends a UTF8_ARRAY field containing <i>utf8_array_data</i> to the end of a message's data. TipcMsgAppendutf8Array makes a copy of the array data.
Caution	None
See Also	TipcMsgAppendUtf8ArrayPtr, TipcMsgNextUtf8Array

Examples This example creates a message and appends a UTF8_ARRAY field:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_STR data[2];

#define USER_MT_UTF8_ARRAY_TEST 100
mt = TipcMtCreate("utf8_array_test", USER_MT_UTF8_ARRAY_TEST,
                 "utf8_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

data[0] = "Hello ";
data[1] = "World";
if (!TipcMsgAppendutf8Array(msg, data, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```

TipcMsgAppendUtf8ArrayPtr

Name	TipcMsgAppendUtf8ArrayPtr — use a pointer to append a field containing an array of UTF8 fields to a message
Synopsis	<pre>T_BOOL TipcMsgAppendUtf8ArrayPtr(msg, utf8_array_data, array_size, field_return) T_IPC_MSG msg; T_STR *utf8_array_data; T_INT4 array_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>utf8_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>utf8_array_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendUtf8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: - T_ERR_NULL_PTR — <i>msg</i> was null or <i>utf8_array_data</i> was null - T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. - T_ERR_MSG_INVALID — <i>msg</i> was not a valid message - T_ERR_VAL_TOO_SMALL — <i>array_size</i> was negative
Description	TipcMsgAppendUtf8ArrayPtr appends a UTF8_ARRAY field containing <i>utf8_array_data</i> to the end of a message’s data. This function does not make a copy of the array. Instead, the supplied pointer to the array is entered directly into the new message field’s internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendUtf8ArrayPtr over TipcMsgAppendUtf8Array is that the character strings may be modified after the field is appended, but before the message is sent. Also, the overhead of one copy operation is avoided, because the array is still copied into the connection message buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateUtf8ArrayPtr function.

Caution If the *field_return* argument is used to gain access to the message field, and the size of the field is subsequently modified using *TipcMsgFieldSetSize*, the *size* argument must be specified in elements rather than bytes.

See Also *TipcMsgAppendUtf8Array*, *TipcMsgNextUtf8Array*, *TipcMsgFieldSetSize*, *TipcMsgFieldUpdateUtf8ArrayPtr*

Examples This example creates a message and appends a pointer to a UTF8_ARRAY field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR data[10];
T_IPC_MSG_FIELD field;

#define USER_MT_UTF8_ARRAY_TEST 100
mt = TipcMtCreate("utf8_array_test", USER_MT_UTF8_ARRAY_TEST,
                 "utf8_array");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt)
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendutf8ArrayPtr(msg, data, 0, &field)) {
    return; /* error */
}

data[0] = "Hello ";
data[1] = "World";
if (!TipcMsgFieldSetSize(field, 2)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```


TipcMsgAppendUtf8Ptr

Name	TipcMsgAppendUtf8Ptr — use a pointer to append a UTF8 field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendUtf8Ptr(msg, utf8_data, utf8_size, field_return) T_IPC_MSG msg; T_PTR utf8_data; T_INT4 utf8_size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>utf8_data</i> — data for new message field <i>utf8_size</i> — number of bytes in <i>utf8_data</i> <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendUtf8Ptr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>utf8_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_TOO_SMALL — <i>utf8_size</i> was negative
Description	TipcMsgAppendUtf8Ptr appends a UTF8 field containing <i>utf8_data</i> to the end of a message's data. TipcMsgAppendUtf8Ptr does not make a copy of the UTF8 data. Instead, the supplied pointer to the data is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendUtf8Ptr over TipcMsgAppendUtf8 is that the UTF8 data can be modified after the field is appended, but before the message is sent. Another advantage is that the overhead of one copy operation is avoided. The data is still copied into the connection write buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateUtf8Ptr function.
Caution	None

See Also `TipcMsgAppendUtf8`, `TipcMsgNextUtf8`

Examples This example creates a message and appends a pointer to a UTF8 field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR utf8[80];
T_IPC_MSG_FIELD field;

#define USER_MT_STR_TEST 100
mt = TipcMtCreate("str_test", USER_MT_STR_TEST, "str");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (mt == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

str[0] = '\0';
if (!TipcMsgAppendUtf8Ptr(msg, str, &field)) {
    return; /* error */
}

strcpy(str, "This is the field data.");
if (!TipcMsgFieldSetSize(field, strlen(str) + 1)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */
```

TipcMsgAppendXml

Name	TipcMsgAppendXml — append an XML field to a message
Synopsis	<pre>T_BOOL TipcMsgAppendXml(msg, xml_data) T_IPC_MSG msg; T_XML xml_data;</pre>
Arguments	<i>msg</i> — message to which to append the field <i>xml_data</i> — data for new message field
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendXml fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>xml_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendXml appends an XML object field to the end of a message's data. TipcMsgAppendXml makes a copy of the XML object.
Caution	None
See Also	TipcMsgNextXml

Examples This example creates a message and appends an XML field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_XML xml;

T_XML xml_val;
#define USER_MT_XML_TEST 100
mt = TipcMtCreate("xml_test", USER_MT_XML_TEST, "xml");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}
xml = TutXmlCreate("<data><data1><temp=\"hot\"/></data1></data>");
if (xml == NULL) {
    return; /* error */
}
if (!TipcMsgAppendXml(msg, xml)) {
    return; /* error */
}
```

TipcMsgAppendXmlPtr

Name	TipcMsgAppendXmlPtr — append an XML pointer to a message
Synopsis	<pre>T_BOOL TipcMsgAppendXmlPtr(msg, xml_data, field_return) T_IPC_MSG msg; T_XML xml_data; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append the field to <i>xml_data</i> — XML object for new field <i>field_return</i> — optional location to return field (use NULL if not needed)
Return Values	TRUE if the field was successfully appended to the message, FALSE otherwise.
Diagnostics	If TipcMsgAppendXmlPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>xml_data</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgAppendXmlPtr appends an XML object to the end of a message's data. This function does not make a copy of the string. Instead, the supplied pointer to the string is entered directly into the new message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed. One advantage of TipcMsgAppendXmlPtr over TipcMsgAppendXml is that the string can be modified after the field is appended, but before the message is sent. Another advantage is that the overhead of one copy operation is avoided. The character string is still copied into the connection message buffer when the message is sent. To update the pointer to point to a new memory location, use the TipcMsgFieldUpdateXmlPtr function.
Caution	Caller must ensure that the pointer remains valid until the message is destroyed.
See Also	TipcMsgAppendXml, TipcMsgFieldSetSize, TipcMsgFieldUpdateXmlPtr

Examples This example creates a message and appends a pointer to an XML field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_XML xml;

#define USER_MT_XML_TEST 100
mt = TipcMtCreate("xml_test", USER_MT_XML_TEST, "xml");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (mt == NULL) {
    return; /* error */
}
xml = TutXmlCreate("<data><data1><temp=\"hot\" /></data1></data>");
if (xml == NULL) {
    return; /* error */
}
if (!TipcMsgAppendXmlPtr(msg, xml, NULL)) {
    return; /* error */
}
```

TipcMsgClone - TipcMsgIncrRefCount

This chapter covers these TipcMsg API functions:

- *TipcMsgClone*
- *TipcMsgCreate*
- *TipcMsgDeleteCurrent*
- *TipcMsgDeleteField*
- *TipcMsgDeleteNamedField*
- *TipcMsgDestroy*
- *TipcMsgExistsNamed*
- *TipcMsgFieldSetSize*
- *TipcMsgFieldUpdateBinaryPtr*
- *TipcMsgFieldUpdateBoolArrayPtr*
- *TipcMsgFieldUpdateInt2ArrayPtr*
- *TipcMsgFieldUpdateInt4ArrayPtr*
- *TipcMsgFieldUpdateInt8ArrayPtr*
- *TipcMsgFieldUpdateMsgArrayPtr*
- *TipcMsgFieldUpdateMsgPtr*
- *TipcMsgFieldUpdateReal4ArrayPtr*
- *TipcMsgFieldUpdateReal8ArrayPtr*
- *TipcMsgFieldUpdateReal16ArrayPtr*
- *TipcMsgFieldUpdateStrArrayPtr*
- *TipcMsgFieldUpdateStrPtr*
- *TipcMsgFieldUpdateTimestampArrayPtr*
- *TipcMsgFieldUpdateUtf8ArrayPtr*
- *TipcMsgFieldUpdateUtf8Ptr*
- *TipcMsgFieldUpdateXmlPtr*
- *TipcMsgFileCreate*

- *TpcMsgFileCreateFromFile*
- *TpcMsgFileDestroy*
- *TpcMsgFileRead*
- *TpcMsgFileWrite*
- *TpcMsgGenerateMessageId*
- *TpcMsgGetArrivalTimestamp*
- *TpcMsgGetCompression*
- *TpcMsgGetCorrelationId*
- *TpcMsgGetCurrent*
- *TpcMsgGetCurrentFieldKnown*
- *TpcMsgGetDeliveryMode*
- *TpcMsgGetDeliveryTimeout*
- *TpcMsgGetDest*
- *TpcMsgGetExpiration*
- *TpcMsgGetHeaderStrEncode*
- *TpcMsgGetLbMode*
- *TpcMsgGetMessageId*
- *TpcMsgGetNameCurrent*
- *TpcMsgGetNamedBinary*
- *TpcMsgGetNamedBool*
- *TpcMsgGetNamedBoolArray*
- *TpcMsgGetNamedByte*
- *TpcMsgGetNamedChar*
- *TpcMsgGetNamedInt2*
- *TpcMsgGetNamedInt2Array*
- *TpcMsgGetNamedInt4*
- *TpcMsgGetNamedInt4Array*
- *TpcMsgGetNamedInt8*
- *TpcMsgGetNamedInt8Array*
- *TpcMsgGetNamedMsg*

- *TipcMsgGetNamedMsgArray*
- *TipcMsgGetNamedReal4*
- *TipcMsgGetNamedReal4Array*
- *TipcMsgGetNamedReal8*
- *TipcMsgGetNamedReal8Array*
- *TipcMsgGetNamedReal16*
- *TipcMsgGetNamedReal16Array*
- *TipcMsgGetNamedStr*
- *TipcMsgGetNamedStrArray*
- *TipcMsgGetNamedTimestamp*
- *TipcMsgGetNamedTimestampArray*
- *TipcMsgGetNamedUnknown*
- *TipcMsgGetNamedUtf8*
- *TipcMsgGetNamedUtf8Array*
- *TipcMsgGetNamedXml*
- *TipcMsgGetNumFields*
- *TipcMsgGetPacketSize*
- *TipcMsgGetPriority*
- *TipcMsgGetReadOnly*
- *TipcMsgGetRefCount*
- *TipcMsgGetReplyTo*
- *TipcMsgGetSender*
- *TipcMsgGetSenderTimestamp*
- *TipcMsgGetSeqNum*
- *TipcMsgGetType*
- *TipcMsgGetTypeNamed*
- *TipcMsgGetTypeNum*
- *TipcMsgGetUserProp*
- *TipcMsgIncrRefCount*

TipcMsgClone

Name	TipcMsgClone — make an identical copy of a message
Synopsis	<pre>T_IPC_MSG TipcMsgClone(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to copy
Return Values	Copy of original message if successful, NULL otherwise.
Diagnostics	If TipcMsgClone fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgClone makes a copy of a message. A new message is created, and all properties of the original message are copied to the new message. The copy does not share any memory with the original message. The copy is not marked read-only, even if the original message is.
Caution	When the message returned by TipcMsgClone is no longer needed, destroy it by calling TipcMsgDestroy, unless the message is passed to another function which takes responsibility for destroying the message (such as TipcSrvMsgInsert).
See Also	TipcMsgCreate, TipcMsgDestroy

Examples This example creates a message, appends two fields, makes a copy of the original message, and destroys the original message:

```
T_IPC_MT mt;
T_IPC_MSG src;
T_IPC_MSG dest;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

src = TipcMsgCreate(mt);
if (src == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(src, "speed_limit", 55.0)) {
    return; /* error */
}

dest = TipcMsgClone(src);
if (dest == NULL) {
    return; /* error */
}

if (!TipcMsgDestroy(src)) {
    return; /* error */
}
```

TipcMsgCreate

Name	TipcMsgCreate — create a new message
Synopsis	<pre>T_IPC_MSG TipcMsgCreate(mt) T_IPC_MT mt;</pre>
Arguments	<i>mt</i> — message type
Return Values	New message if successful, NULL otherwise.
Diagnostics	If TipcMsgCreate fails, it returns NULL and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>mt</i> was null
Description	TipcMsgCreate creates a new message with the desired type, with these properties: • The new message has a sender property of NULL, a destination property of NULL, zero fields (that is, the data property is empty), and a reference count of 1. • The message’s priority property is initialized to the message type priority (if set) or to the value of the option Default_Msg_Priority (if the message type priority is unknown). • The message’s delivery mode property is initialized to the message type delivery mode. • The message’s delivery timeout property is initialized to the message type delivery timeout. • The message’s load balancing mode property is initialized to the message type load balancing mode. • The message’s header string encode property is initialized to the message type header string encode. • The message’s user-defined property is initialized to the message type user-defined property. • The message’s data buffer is created with enough storage for 1024 bytes of data. Fields can be added to the message with the TipcMsgAppend* functions.
Caution	When the message returned by TipcMsgCreate is no longer needed, destroy it by calling TipcMsgDestroy, unless the message is passed to another function which takes responsibility for destroying the message, such as TpcSrvMsgInsert.

See Also `TipcMsgDestroy`

Examples This example creates a message, appends two fields, and destroys the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendStrReal8(msg, "speed_limit", 55.0)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcMsgDeleteCurrent

Name	TipcMsgDeleteCurrent — delete the current field from a message
Synopsis	<pre>T_BOOL TipcMsgDeleteCurrent(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to delete field from
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgDeleteCurrent fails, it returns FALSE and sets the global SmartSockets error numbers to one of the following: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgDeleteCurrent deletes the current field from the message, whether the field is named or unnamed. If no such field exists, it returns FALSE.
See Also	TipcMsgDeleteNamedField, TipcMsgDeleteField
Examples	This code example deletes the current field from the message: <pre>if (!TipcMsgDeleteCurrent(msg)) { return; /* error */ }</pre>

TipcMsgDeleteField

Name	TipcMsgDeleteField — delete a field, specified by field number, from a message
Synopsis	<pre>T_BOOL TipcMsgDeleteField(msg, field_num) T_IPC_MSG msg; T_INT4 field_num;</pre>
Arguments	<i>msg</i> — message to delete field from <i>field_num</i> — number of the field
Return Values	TRUE if the deletion was successful, FALSE otherwise.
Diagnostics	If TipcMsgDeleteField fails, it returns FALSE and sets the global SmartSockets error numbers to one of the following: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgDeleteField deletes the field of the specified number from the message, irrespective of whether the field is named or unnamed, and updates the field numbers of the subsequent fields. (For example, if field 3 is deleted, field 4 becomes field 3, field 5 becomes field 4, and so on.) If no such field exists, the function returns FALSE.
See Also	TipcMsgDeleteNamedField, TipcMsgDeleteCurrent
Examples	This code example deletes the field of the specified number from the message: <pre>if (!TipcMsgDeleteField(msg, field_num)) { return; /* error */ }</pre>

TipcMsgDeleteNamedField

Name	TipcMsgDeleteNamedField — delete a field, specified by field name, from a message
Synopsis	<pre>T_BOOL TipcMsgDeleteNamedField(msg, name) T_IPC_MSG msg; T_STR name;</pre>
Arguments	<i>msg</i> — message to delete field from <i>name</i> — name of the field to delete
Return Values	TRUE if the deletion was successful, FALSE otherwise.
Diagnostics	If TipcMsgDeleteNamedField fails, it returns FALSE and sets the global SmartSockets error numbers to one of the following: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgDeleteNamedField deletes the named field from the message. The index positions associated with the subsequent named fields are updated. If no such field exists, the function returns FALSE.
See Also	TipcMsgDeleteCurrent, TipcMsgDeleteField
Examples	This example deletes the named field "cost": <pre>if (!TipcMsgDeleteNamedField(msg, "cost")) { return; /* error */ }</pre>

TipcMsgDestroy

Name	TipcMsgDestroy — destroy a message
Synopsis	<pre>T_BOOL TipcMsgDestroy(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to destroy
Return Values	TRUE if the message was successfully destroyed, FALSE otherwise.
Diagnostics	If TipcMsgDestroy fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgDestroy decrements the reference count of a message and destroys the message if the reference count is zero. When a message is destroyed, all of its memory is deallocated and the message cannot be reused. If the message's delivery mode is either T_IPC_DELIVERY_SOME or T_IPC_DELIVERY_ALL and the message has not already been acknowledged manually with TipcMsgAck, then TipcMsgDestroy first calls TipcMsgAck to acknowledge successful delivery of the message to this receiving process. TipcMsgDestroy next decrements the reference count of the message, and if the reference count is still greater than zero, TipcMsgDestroy immediately returns. All memory for the message is then freed.
Caution	TipcMsgAck sends a GMD_ACK message, but does not explicitly flush the message. See TipcConnGetAutoFlushSize for more information on message buffering.
See Also	TipcMsgCreate, TipcMsgAck

Examples This example creates a message, appends two fields, and destroys the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgAppendStrReal8(msg, "speed_limit", 55.0)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcMsgExistsNamed

Name	TipcMsgExistsNamed — determines if a field exists in a message, using a name
Synopsis	<pre>T_BOOL TipcMsgExistsNamed(msg, name, exists) T_IPC_MSG msg; T_STR name; T_BOOL *exists;</pre>
Arguments	<i>msg</i> — message to check <i>name</i> — name of the field <i>exists</i> — pointer to a boolean indicating whether or not the field exists
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	None
Description	TipcMsgExistsNamed checks to see if the field indicated by the name exists in the message. The exists return value is set to TRUE if the field exists, FALSE otherwise.
Examples	This example checks to see if a field named "foo" exists in the message: <pre>T_BOOL status; T_BOOL exists; status = TipcMsgExistsNamed(msg, "foo", &exists); if (status == FALSE) { return; /* error */ } if (exists == TRUE) { /* field exists */ }</pre>

TipcMsgFieldSetSize

Name	TipcMsgFieldSetSize — set the size of a message pointer field
Synopsis	<pre>T_BOOL TipcMsgFieldSetSize(<i>field</i>, <i>size</i>) T_IPC_MSG_FIELD <i>field</i>; T_INT4 <i>size</i>;</pre>
Arguments	<i>field</i> — message pointer field to resize <i>size</i> — new size (usually in bytes)
Return Values	TRUE if field size set successfully, FALSE if error occurred.
Diagnostics	If TipcMsgFieldSetSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>field</i> was null • T_ERR_VAL_TOO_SMALL — <i>size</i> was negative
Description	TipcMsgFieldSetSize sets the size of a message pointer field (one created with a TipcMsgAppend*Ptr function). For scalar and binary field types, the <i>size</i> argument is specified in bytes. For all array field types (including MSG_ARRAY), the <i>size</i> argument is specified in elements. For individual MSG fields, the <i>size</i> argument must always be zero.
Caution	None
See Also	The TipcMsgAppend*Ptr functions
Examples	This example sets the size of a message pointer field: <pre>T_CHAR str[80]; T_IPC_MSG_FIELD field; str[0] = '\0'; if (!TipcMsgAppendStrPtr(msg, str, &field)) { return; /* error */ } strcpy(str, "This is the field data."); if (!TipcMsgFieldSetSize(field, strlen(str) + 1)) { return; /* error */ }</pre>

TipcMsgFieldUpdateBinaryPtr

Name	TipcMsgFieldUpdateBinaryPtr — update the pointer of the message field returned by TipcMsgAppendBinaryPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateBinaryPtr(<i>field</i>, <i>binary_data</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_PTR <i>binary_data</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>binary_data</i> — data for new message field <i>array_size</i> — number of elements in <i>binary_data</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateBinaryPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateBinaryPtr updates the pointer of the message field returned by TipcMsgAppendBinaryPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendBinaryPtr
Examples	This example changes the pointer of the BINARY field: <pre>T_PTR ptr1, ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendBinaryPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateBinaryPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateBoolArrayPtr

Name	TipcMsgFieldUpdateBoolArrayPtr — update the pointer of the message field returned by TipcMsgAppendBoolArrayPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateBoolArrayPtr(<i>field</i>, <i>bool_array</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_BOOL *<i>bool_array</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>bool_array</i> — data for new message field <i>array_size</i> — number of elements in <i>bool_array</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateBoolArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateBoolArrayPtr updates the pointer of the message field returned by TipcMsgAppendBoolArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendBoolArrayPtr
Examples	This example changes the pointer of the BOOL_ARRAY field: <pre>T_BOOL *ptr1, *ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendBoolArrayPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateBoolArrayPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateInt2ArrayPtr

Name	TipcMsgFieldUpdateInt2ArrayPtr — update the pointer of the message field returned by TipcMsgAppendInt2ArrayPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateInt2ArrayPtr(<i>field</i>, <i>*int2_array</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_INT2 <i>*int2_array</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>int2_array</i> — data for new message field <i>array_size</i> — number of elements in <i>int2_array</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateInt2ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateInt2ArrayPtr updates the pointer of the message field returned by TipcMsgAppendInt2ArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendInt2ArrayPtr
Examples	This example changes the pointer of the INT2_ARRAY field: <pre>T_INT2 *ptr1, *ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendInt2ArrayPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateInt2ArrayPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateInt4ArrayPtr

Name	TipcMsgFieldUpdateInt4ArrayPtr — update the pointer of the message field returned by TipcMsgAppendInt4ArrayPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateInt4ArrayPtr(<i>field</i>, <i>*int4_array</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_INT4 <i>*int4_array</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>int4_array</i> — data for new message field <i>array_size</i> — number of elements in <i>int4_array</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateInt4ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateInt4ArrayPtr updates the pointer of the message field returned by TipcMsgAppendInt4ArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendInt4ArrayPtr
Examples	This example changes the pointer of the INT4_ARRAY field: <pre>T_INT4 *ptr1, *ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendInt4ArrayPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateInt4ArrayPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateInt8ArrayPtr

Name	TipcMsgFieldUpdateInt8ArrayPtr — update the pointer of the message field returned by TipcMsgAppendInt8ArrayPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateInt8ArrayPtr(<i>field</i>, <i>*int8_array</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_INT8 <i>*int8_array</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>int8_array</i> — data for new message field <i>array_size</i> — number of elements in <i>int8_array</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateInt8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateInt8ArrayPtr updates the pointer of the message field returned by TipcMsgAppendInt8ArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendInt8ArrayPtr
Examples	This example changes the pointer of the INT8_ARRAY field: <pre>T_INT8 *ptr1, *ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendInt8ArrayPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateInt8ArrayPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TpcMsgFieldUpdateMsgArrayPtr

Name	TpcMsgFieldUpdateMsgArrayPtr — update the pointer of the message field returned by TpcMsgAppendMsgArrayPtr
Synopsis	<pre>T_BOOL TpcMsgFieldUpdateMsgArrayPtr(<i>field</i>, *<i>msg_array</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_MSG *<i>msg_array_data</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>msg_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>msg_array_data</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TpcMsgFieldUpdateMsgArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TpcMsgFieldUpdateMsgArrayPtr updates the pointer of the message field returned by TpcMsgAppendMsgArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TpcMsgAppendMsgArrayPtr
Examples	This example changes the pointer of the MSG_ARRAY field: <pre>T_IPC_MSG *ptr1, *ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TpcMsgAppendMsgArrayPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TpcMsgFieldUpdateMsgArrayPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateMsgPtr

Name	TipcMsgFieldUpdateMsgPtr — update the pointer of the message field returned by TipcMsgAppendMsgPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateMsgPtr(<i>field</i>, <i>msg_data</i>) T_IPC_MSG_FIELD <i>field</i>; T_MSG <i>msg_data</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>msg_data</i> — data for new message field
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateMsgPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: - T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message - T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateMsgPtr updates the pointer of the message field returned by TipcMsgAppendMsgPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendMsgPtr
Examples	This example changes the pointer of the MSG field: <pre>T_IPC_MSG ptr1, ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendMsgPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateMsgPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateReal4ArrayPtr

Name	TipcMsgFieldUpdateReal4ArrayPtr — update the pointer of the message field returned by TipcMsgAppendReal4ArrayPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateReal4ArrayPtr(<i>field</i>, <i>*real4_array</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_REAL4 <i>*real4_array</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>real4_array</i> — data for new message field <i>array_size</i> — number of elements in <i>real4_array</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateReal4ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateReal4ArrayPtr updates the pointer of the message field returned by TipcMsgAppendReal4ArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendReal4ArrayPtr
Examples	This example changes the pointer of the REAL4_ARRAY field: <pre>T_REAL4 *ptr1, *ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendReal4ArrayPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateReal4ArrayPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateReal8ArrayPtr

Name	TipcMsgFieldUpdateReal8ArrayPtr — update the pointer of the message field returned by TipcMsgAppendReal8ArrayPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateReal8ArrayPtr(<i>field</i>, <i>*real8_array</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_REAL8 <i>*real8_array</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>real8_array</i> — data for new message field <i>array_size</i> — number of elements in <i>real8_array</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateReal8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateReal8ArrayPtr updates the pointer of the message field returned by TipcMsgAppendReal8ArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendReal8ArrayPtr
Examples	This example changes the pointer of the REAL8_ARRAY field: <pre>T_REAL8 *ptr1, *ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendReal8ArrayPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateReal8ArrayPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateReal16ArrayPtr

Name	TipcMsgFieldUpdateReal16ArrayPtr — update the pointer of the message field returned by TipcMsgAppendReal16ArrayPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateReal16ArrayPtr(<i>field</i>, *<i>real16_array</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_REAL16 *<i>real16_array</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>real16_array</i> — data for new message field <i>array_size</i> — number of elements in <i>real16_array</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateReal16ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: - T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message - T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateReal16ArrayPtr updates the pointer of the message field returned by TipcMsgAppendReal16ArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
	Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.
See Also	TipcMsgAppendReal16ArrayPtr

Examples This example changes the pointer of the REAL16_ARRAY field:

```
T_REAL16 *ptr1, *ptr2;
T_INT4 size1, size2;
T_IPC_MSG_FIELD field;

if (!TipcMsgAppendReal16ArrayPtr(msg, ptr1, size1, &field)) {
    return; /* error */
}

if (!TipcMsgFieldUpdateReal16ArrayPtr(field, ptr2, size2))
    return; /* error */
}
```

TipcMsgFieldUpdateStrArrayPtr

Name	TipcMsgFieldUpdateStrArrayPtr — update the pointer of the message field returned by TipcMsgAppendStrArrayPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateStrArrayPtr(<i>field</i>, <i>*str_array_data</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_STR <i>*str_array_data</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>str_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>str_array_data</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateStrArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateStrArrayPtr updates the pointer of the message field returned by TipcMsgAppendStrArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendStrArrayPtr
Examples	This example changes the pointer of the STR_ARRAY field: <pre>T_STR *ptr1, *ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendStrArrayPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateStrArrayPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateStrPtr

Name	TipcMsgFieldUpdateStrPtr — update the pointer of the message field returned by TipcMsgAppendStrPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateStrPtr(<i>field</i>, <i>str_data</i>) T_IPC_MSG_FIELD <i>field</i>; T_STR <i>str_data</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>str_data</i> — data for new message field
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateStrPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateStrPtr updates the pointer of the message field returned by TipcMsgAppendStrPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendStrPtr
Examples	This example changes the pointer of the STR field: <pre>T_STR ptr1, ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendStrPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateStrPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateTimestampArrayPtr

Name	TipcMsgFieldUpdateTimestampArrayPtr — update the pointer of the message field returned by TipcMsgAppendTimestampArrayPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateTimestampArrayPtr(<i>field</i>, <i>*timestamp_array</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_REAL8 <i>*timestamp_array</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>timestamp_array</i> — data for new message field <i>array_size</i> — number of elements in <i>timestamp_array</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateTimestampArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateTimestampArrayPtr updates the pointer of the message field returned by TipcMsgAppendTimestampArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendTimestampArrayPtr
Examples	This example changes the pointer of the <code>TIMESTAMP_ARRAY</code> field: <pre>T_REAL8 *ptr1, *ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendTimestampArrayPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateTimestampArrayPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateUtf8ArrayPtr

Name	TipcMsgFieldUpdateUtf8ArrayPtr — update the pointer of the message field returned by TipcMsgAppendUtf8ArrayPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateUtf8ArrayPtr(<i>field</i>, *<i>utf8_array_data</i>, <i>array_size</i>) T_IPC_MSG_FIELD <i>field</i>; T_STR *<i>utf8_array_data</i>; T_INT4 <i>array_size</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>utf8_array_data</i> — data for new message field <i>array_size</i> — number of elements in <i>utf8_array_data</i>
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateUtf8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateUtf8ArrayPtr updates the pointer of the message field returned by TipcMsgAppendUtf8ArrayPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendUtf8ArrayPtr
Examples	This example changes the pointer of the UTF8_ARRAY field: <pre>T_STR *ptr1, *ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendUtf8ArrayPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateUtf8ArrayPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateUtf8Ptr

Name	TipcMsgFieldUpdateUtf8Ptr — update the pointer of the message field returned by TipcMsgAppendUtf8Ptr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateUtf8Ptr(<i>field</i>, <i>utf8_data</i>) T_IPC_MSG_FIELD <i>field</i>; T_STR <i>utf8_data</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>utf8_data</i> — data for new message field
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateUtf8Ptr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateUtf8Ptr updates the pointer of the message field returned by TipcMsgAppendUtf8Ptr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendUtf8Ptr
Examples	This example changes the pointer of the UTF8 field: <pre>T_STR ptr1, ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendUtf8Ptr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateUtf8Ptr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFieldUpdateXmlPtr

Name	TipcMsgFieldUpdateXmlPtr — update the pointer of the message field returned by TipcMsgAppendXmlPtr
Synopsis	<pre>T_BOOL TipcMsgFieldUpdateXmlPtr(<i>field</i>, <i>xml_data</i>) T_IPC_MSG_FIELD <i>field</i>; T_XML <i>xml_data</i>;</pre>
Arguments	<i>field</i> — message pointer field to update <i>xml_data</i> — data for new message field
Return Values	TRUE if the pointer was successfully updated, FALSE otherwise.
Diagnostics	If TipcMsgFieldUpdateXmlPtr fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgFieldUpdateXmlPtr updates the pointer of the message field returned by TipcMsgAppendXmlPtr. The contents of the pointer are not copied. Instead, the supplied pointer is entered directly into the message field's internal data structure. The caller is responsible for making sure that the pointer remains valid for the lifetime of the message field, and for freeing it, if necessary, when the message field has been destroyed.
See Also	TipcMsgAppendXmlPtr
Examples	This example changes the pointer of the XML field: <pre>T_XML ptr1, ptr2; T_INT4 size1, size2; T_IPC_MSG_FIELD field; if (!TipcMsgAppendXmlPtr(msg, ptr1, size1, &field)) { return; /* error */ } if (!TipcMsgFieldUpdateXmlPtr(field, ptr2, size2)) { return; /* error */ }</pre>

TipcMsgFileCreate

Name	TipcMsgFileCreate — create a message file
Synopsis	<pre>T_IPC_MSG_FILE TipcMsgFileCreate(<i>file_name</i>, <i>create_mode</i>) T_STR <i>file_name</i>; T_IPC_MSG_FILE_CREATE_MODE <i>create_mode</i>;</pre>
Arguments	<i>file_name</i> — name of file to use for message file <i>create_mode</i> — how to create the file. Use one of these modes: • T_IPC_MSG_FILE_CREATE_READ creates the message file in an existing file • T_IPC_MSG_FILE_CREATE_WRITE for writing • T_IPC_MSG_FILE_CREATE_WRITE_BINARY for writing a binary file • T_IPC_MSG_FILE_CREATE_APPEND for appending
Return Values	A message file if successful, NULL otherwise.
Diagnostics	If TipcMsgFileCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>file_name</i> was null • T_ERR_VAL_INVALID — <i>create_mode</i> was not a valid T_IPC_MSG_FILE_CREATE_MODE create mode • T_ERR_C — a C error occurred. Call the function TutErrNumGetC to obtain the related C-specific error value. • T_ERR_MSG_CORRUPT — the message file is not in the proper format • T_ERR_TYPE_INVALID — the message file is in binary format but was created on an architecture that is incompatible with this one
Description	TipcMsgFileCreate creates a new message file. There are two possible formats for message files: • an architecture-independent text format which is slower to read and write but can be edited with any text editor • an architecture-dependent binary format which is faster to read and write but cannot be easily edited with a text editor

If *create_mode* is `T_IPC_MSG_FILE_CREATE_READ`, *file_name* must be the name of an existing file. The message file is opened for reading in an existing file, and `TipcMsgFileRead` can be used to read messages from the message file. `TipcMsgFileCreate` automatically tries to detect whether the file is in text or binary format.

If *create_mode* is `T_IPC_MSG_FILE_CREATE_WRITE`, the message file is created for writing in text format, and `TipcMsgFileWrite` can be used to write messages to the message file. If *file_name* already exists, it is overwritten. If *file_name* does not already exist, a file is created.

If *create_mode* is `T_IPC_MSG_FILE_CREATE_WRITE_BINARY`, the message file is created for writing in binary format, and `TipcMsgFileWrite` can be used to write messages to the message file. If *file_name* already exists, it is overwritten. If *file_name* does not already exist, a file is created.

If *create_mode* is `T_IPC_MSG_FILE_CREATE_APPEND`, the message file is opened for appending, and `TipcMsgFileWrite` can be used to append messages to the message file. `TipcMsgFileCreate` automatically tries to detect whether the file is in text or binary format. If *file_name* already exists, the existing file is appended to. If *file_name* does not already exist, a file is created.

To create a message file from an existing file (such as `stdin`, `stdout`, or `stderr`), use `TipcMsgFileCreateFromFile`.

Binary message files can be shared between dissimilar architectures (from Digital Alpha AXP to Sun SPARC, for example), and `TipcMsgFileCreate` automatically converts the file data.

Caution If *create_mode* is `T_IPC_MSG_FILE_CREATE_WRITE` or `T_IPC_MSG_FILE_CREATE_WRITE_BINARY`, any existing file is overwritten.

If *create_mode* is `T_IPC_MSG_FILE_CREATE_READ`, the function is trying to read to an existing file. If there is no existing file, the function fails and returns an error.

See Also `TipcMsgFileCreateFromFile`, `TipcMsgFileDestroy`, `TipcMsgFileRead`, `TipcMsgFileWrite`

Examples This example creates a message file for reading from the existing file `data.msg` with `TipcMsgFileCreate`:

```
T_IPC_MSG_FILE msg_file;

msg_file = TipcMsgFileCreate("data.msg",
T_IPC_MSG_FILE_CREATE_READ);
if (msg_file == NULL) {
    return; /* error */
}
```

TipcMsgFileCreateFromFile

Name	TipcMsgFileCreateFromFile — create a message file from an existing file
Synopsis	<pre>T_IPC_MSG_FILE TipcMsgFileCreateFromFile(<i>file</i>, <i>create_mode</i>) FILE *<i>file</i>; T_IPC_MSG_FILE_CREATE_MODE <i>create_mode</i>;</pre>
Arguments	<i>file</i> — existing file to use for message file <i>create_mode</i> — how to create the file
Return Values	A new message file if successful, NULL otherwise.
Diagnostics	If TipcMsgFileCreateFromFile fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>file</i> was null • T_ERR_VAL_INVALID — <i>create_mode</i> was not a valid T_IPC_MSG_FILE_CREATE_MODE create mode • T_ERR_C — a C error has occurred. Call the function TutErrNumGetC to obtain the related C-specific error value. • T_ERR_MSG_CORRUPT — the message file is not in the proper format
Description	TipcMsgFileCreateFromFile creates a new message file from an existing file. TipcMsgFileCreateFromFile can be used in those cases where a C <code>stdio</code> file is available, but not a file name (<code>stdin</code>, <code>stdout</code>, and <code>stderr</code>, for example). TipcMsgFileCreateFromFile can only be used with text format message files. See TipcMsgFileCreate for more information on text format versus binary format. If <i>create_mode</i> is: • T_IPC_MSG_FILE_CREATE_READ, then the message file is created for reading, and TipcMsgFileRead can be used to read messages from the message file. • T_IPC_MSG_FILE_CREATE_WRITE, then the message file is created for writing, and TipcMsgFileWrite can be used to write messages to the message file. • T_IPC_MSG_FILE_CREATE_APPEND, then the message file is created for appending, and TipcMsgFileWrite can be used to append messages to the message file.
Caution	None

See Also TipcMsgFileCreate, TipcMsgFileDestroy, TipcMsgFileRead, TipcMsgFileWrite

Examples This example creates a message file for writing from the file `stdout` with `TipcMsgFileCreateFromFile`:

```
T_IPC_MSG_FILE msg_file;

msg_file = TipcMsgFileCreateFromFile(stdout,
T_IPC_MSG_FILE_CREATE_WRITE);
if (msg_file == NULL) {
    return; /* error */
}
```

TipcMsgFileDestroy

Name TipcMsgFileDestroy — destroy a T_IPC_MSG_FILE

Synopsis `T_BOOL TipcMsgFileDestroy(msg_file)
T_IPC_MSG_FILE msg_file;`

Arguments *msg_file* — T_IPC_MSG_FILE to destroy

Return Values TRUE if the T_IPC_MSG_FILE was successfully destroyed, FALSE otherwise.

Diagnostics If TipcMsgFileDestroy fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *msg_file* was null

Description TipcMsgFileDestroy destroys a T_IPC_MSG_FILE. All memory for the T_IPC_MSG_FILE is freed. If the message file was created with TipcMsgFileCreateFromFile, then the file is left open, otherwise the file is closed. The file on disk is not removed.

Caution TipcMsgFileDestroy does not remove the message file from disk.

See Also TipcMsgFileCreate, TipcMsgFileRead, TipcMsgFileWrite

Examples This example destroys a T_IPC_MSG_FILE with TipcMsgFileDestroy:

```
if (!TipcMsgFileDestroy(msg_file)) {  
    return; /* error */  
}
```

TipcMsgFileRead

Name	TipcMsgFileRead — read a message from a message file
Synopsis	<pre>T_BOOL TipcMsgFileRead(msg_file, msg_return) T_IPC_MSG_FILE msg_file; T_IPC_MSG *msg_return;</pre>
Arguments	<i>msg_file</i> — message file <i>msg_return</i> — storage for message from file
Return Values	TRUE if a message was successfully read from the message file, FALSE otherwise.
Diagnostics	If TipcMsgFileRead fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg_file</i> was null or <i>msg_return</i> was null • T_ERR_C — a C error has occurred. Call the function TutErrNumGetC to obtain the related C-specific error value. • T_ERR_VAL_INVALID — <i>msg_file</i> was not created with mode T_IPC_MSG_FILE_CREATE_READ • T_ERR_MSG_CORRUPT — <i>msg_file</i> is not in the proper format • T_ERR_EOF — the end of the file was reached
Description	TipcMsgFileRead reads one message from a message file. See TipcMsgFileCreate for more information on message files. For text message files, TipcMsgFileRead reads the message type name, creates a message, sets the sender of the message to <i>_file</i> , reads the destination, sets the destination, and finally reads the fields and appends them to the message.
Caution	When the message returned in <i>msg_return</i> by TipcMsgFileRead is no longer needed, destroy it by calling TipcMsgDestroy, unless the message is passed to another function which takes responsibility for destroying the message (such as TipcSrvMsgInsert).
See Also	TipcMsgFileCreate, TipcMsgFileCreateFromFile, TipcMsgFileDestroy, TipcMsgFileWrite

Examples This example creates a message file from the file data.msg, reads all the messages in the file using TipcMsgFileRead, and prints out the message with TipcMsgPrint:

```
T_IPC_MSG_FILE msg_file;
T_IPC_MSG msg;

msg_file = TipcMsgFileCreate("data.msg",
T_IPC_MSG_FILE_CREATE_READ);
if (msg_file == NULL) {
    return; /* error */
}

while (TipcMsgFileRead(msg_file, &msg)) {
    TutOut("Read a message.\n");
    if (!TipcMsgPrint(msg, TutOut)) {
        return; /* error */
    }
    if (!TipcMsgDestroy(msg)) {
        return; /* error */
    }
}

if (!TipcMsgFileDestroy(msg_file)) {
    return; /* error */
}
```

If the file data.msg contained the information:

```
numeric_data thermal {
    bat1_voltage 42.4
}
```

then the output from the above example is similar to:

```
Read a message.
type = numeric_data
sender = <_file>
dest = <thermal>
max = 2048
size = 45
current = 0
read_only = false
priority = 0
delivery_mode = best_effort
ref_count = 1
seq_num = 0
resend_mode = false
user_prop = 0
data (num_fields = 2):
    str "bat1_voltage"
    real8 42.4
```

TipcMsgFileWrite

Name	TipcMsgFileWrite — write a message to a message file
Synopsis	<pre>T_BOOL TipcMsgFileWrite(msg_file, msg) T_IPC_MSG_FILE msg_file; T_IPC_MSG msg;</pre>
Arguments	<i>msg_file</i> — message file <i>msg</i> — message to write to message file
Return Values	TRUE if the message was successfully written to the message file, FALSE otherwise.
Diagnostics	If TipcMsgFileWrite fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg_file</i> was null or <i>msg</i> was null • T_ERR_C — a C error has occurred. Call the function TutErrNumGetC to obtain the related C-specific error value. • T_ERR_VAL_INVALID — <i>msg_file</i> was not created with mode T_IPC_MSG_FILE_CREATE_WRITE, T_IPC_MSG_FILE_CREATE_WRITE_BINARY, or T_IPC_MSG_FILE_CREATE_APPEND • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message

Description TipcMsgFileWrite writes a message to a message file. See TipcMsgFileCreate for more information on message files.

Text Message Files

TipcMsgFileWrite writes the message type, then the destination (or `_null` if the destination is null), and finally writes the fields to the file. The message source and priority are not written to the file. After all output has been written to the file, TipcMsgFileWrite flushes the file to disk.

TipcMsgFileWrite first looks up the message type with TipcMtLookupByNum. If the message type does not exist (this can happen if a process such as RTserver receives a message for which it has not created a message type with TipcMtCreate), then TipcMsgFileWrite writes the message type number (for example, 22); otherwise it writes the message type name (for example, `numeric_data`).

The message fields are written to the file in a verbose format if the message type does not exist or the message type grammar is "verbose". The verbose format uses a type name and value for each field (such as `real8 34.567891`). Arrays are delimited by `{}`.

If the verbose format is not used, then only the value is used for each field (such as `34.567891`). The message type grammar is used to help format the output. Both groups and arrays are delimited by `{}`.

Caution For text message files, TipcMsgFileWrite fails if the fields in the message do not match the grammar of the message type (for example, the number of fields in the grammar is not the same as the number of fields in the message, or the types of the fields in the grammar do not match the types of the fields in the message).

See Also TipcMsgFileRead, TipcMtCreate, TipcMtLookup, TipcMtLookupByNum

Examples This example creates a NUMERIC_DATA message and writes it to the message file data.msg:

```
T_IPC_MSG_FILE msg_file;
T_IPC_MT mt;
T_IPC_MSG msg;

msg_file = TipcMsgFileCreate("data.msg",
T_IPC_MSG_FILE_CREATE_WRITE);
if (msg_file == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgSetDest(msg, "/system/thermal")) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "bat1_voltage", 42.4)) {
    return; /* error */
}
if (!TipcMsgFileWrite(msg_file, msg)) {
    return; /* error */
}
```

The output from this message in data.msg would look like:

```
numeric_data /system/thermal {
    bat1_voltage 42.4000000
}
```

TipcMsgGenerateMessageId

Name	TipcMsgGenerateMessageId — generate the message identifier property for the message
Synopsis	<pre>T_BOOL TipcMsgGenerateMessageId(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to generate identifier for
Return Values	TRUE if the identifier was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgGenerateMessageId fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — message was null • T_ERR_MSG_INVALID — message was not a valid message • T_ERR_READ_ONLY — message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgGenerateMessageId generates the unique message identifier property for the message. Applications of messages are free to use the message identifier for any purpose. This message identifier is unique for all messages that are created anywhere. It is sent along with the message and adds 38 extra bytes to the message.
Caution	None
See Also	TipcMsgGetMessageId, TipcMsgSetCorrelationId, TipcMsgGetCorrelationId
Examples	This example generates a message identifier: <pre>if (!TipcMsgGenerateMessageId(msg)) { return; /* error */ }</pre>

TipcMsgGetArrivalTimestamp

Name	TipcMsgGetArrivalTimestamp — get the arrival timestamp property of a message
Synopsis	<pre>T_BOOL TipcMsgGetArrivalTimestamp(msg, arrival_timestamp_return) T_IPC_MSG msg; T_REAL8 *arrival_timestamp_return;</pre>
Arguments	<i>msg</i> — message to get arrival timestamp from <i>arrival_timestamp_return</i> — storage for message arrival timestamp
Return Values	TRUE if the arrival timestamp was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetArrivalTimestamp fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>arrival_timestamp_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetArrivalTimestamp gets the arrival timestamp property of a message. As messages are read, their arrival timestamp property will automatically be set. A value of 0.0 indicates that the arrival timestamp property was not set.
See Also	TipcMsgSetArrivalTimestamp
Examples	This example gets the arrival timestamp of a message: <pre>T_REAL8 arrival_timestamp; if (!TipcMsgGetArrivalTimestamp(msg, &arrival_timestamp)) { return; /* error */ }</pre>

TipcMsgGetCompression

Name TipcMsgGetCompression — get the compression status of a message

Synopsis `T_BOOL TipcMsgGetCompression(msg, compress_return)`
`T_IPC_MSG msg;`
`T_BOOL *compress_return;`

Arguments *msg* — message to get compression status from
compress_return — storage for message compression status

Return Values TRUE if the compression status was successfully retrieved from the message,
FALSE otherwise.

Diagnostics If TipcMsgGetCompression fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *msg* or *compress_return* was null
- T_ERR_MSG_INVALID — *msg* was not a valid message

Description TipcMsgGetCompression gets the compression status of a message, or whether or not compression is enabled. When a message is created, its compression status is initialized to the message type's compression status. Use TipcMsgSetCompression to change the default compression status of a message.

Caution If TipcMsgGetCompression returns FALSE, it does not store a value in *compress_return*.

See Also TipcMsgSetCompression, TipcMtGetCompression

Examples This example gets the compression status of a message:

```
T_BOOL compress;

if (!TipcMsgGetCompression(msg, &compress)) {
    return; /* error */
}
```

TipcMsgGetCorrelationId

Name	TipcMsgGetCorrelationId — get the correlation identifier property of a message
Synopsis	<pre>T_BOOL TipcMsgGetCorrelationId(msg, correlation_id_return) T_IPC_MSG msg; T_STR *correlation_id_return;</pre>
Arguments	<i>msg</i> — message to get identifier from <i>correlation_id_return</i> — storage for identifier
Return Values	TRUE if the identifier was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetCorrelationId fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>correlation_id_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetCorrelationId gets the correlation identifier property of a message. The identifier property is a string whose meaning depends on how the message is being used. Applications of messages are free to use the correlation identifier for any purpose.
Caution	If TipcMsgGetCorrelationId returns FALSE, it does not store a value in <i>correlation_id_return</i>. Do not modify the value stored in <i>correlation_id_return</i>. It points directly into an internal data structure.
See Also	TipcMsgSetCorrelationId, TipcMsgGenerateMessageId, TipcMsgGetMessageId
Examples	This example prints the correlation identifier of a message: <pre>T_STR id; if (!TipcMsgGetCorrelationId(msg, &id)) { return; /* error */ } TutOut("msg had identifier %s\n", id ? id : "<NULL>");</pre>

TipcMsgGetCurrent

Name	TipcMsgGetCurrent — get the current field of a message
Synopsis	<pre>T_BOOL TipcMsgGetCurrent(msg, field_num_return) T_IPC_MSG msg; T_INT4 *field_num_return;</pre>
Arguments	<i>msg</i> — message to get current field from <i>field_num_return</i> — storage for message field number
Return Values	TRUE if the current field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetCurrent fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>field_num_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is past the last field of the message
Description	TipcMsgGetCurrent gets the current field of a message as a field number between 0 and the number of fields in the message minus one.
Caution	If TipcMsgGetCurrent returns FALSE, it does not store a value in <i>field_num_return</i> .
See Also	TipcMsgGetNumFields, TipcMsgSetCurrent

Examples This example creates a message, appends fields, gets the current field, reads the fields, and gets the current field:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_STR str;
T_REAL8 real8;
T_INT4 field_num;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "speed_limit", 65.0)) {
    return; /* error */
}

if (!TipcMsgGetCurrent(msg, &field_num)) {
    return; /* error */
}
TutOut("current field is %d\n", field_num);

if (!TipcMsgNextStr(msg, &str)) {
    return; /* error */
}
TutOut("str = %s\n", str);

if (!TipcMsgGetCurrent(msg, &field_num)) {
    return; /* error */
}
TutOut("current field is %d\n", field_num);

if (!TipcMsgNextReal8(msg, &real8)) {
    return; /* error */
}
TutOut("real8 = %s\n", TutRealToStr(real8));

if (!TipcMsgGetCurrent(msg, &field_num)) {
    return; /* error */
}
TutOut("current field is %d\n", field_num);

```

TipcMsgGetCurrentFieldKnown

Name	TipcMsgGetCurrentFieldKnown — checks if the current field in a message is of known value
Synopsis	<pre>T_BOOL TipcMsgGetCurrentFieldKnown(msg, field_valid_return) T_IPC_MSG msg; T_BOOL *field_valid_return;</pre>
Arguments	<i>msg</i> — message to get status of the current field from <i>field_valid_return</i> — storage for status
Return Values	TRUE if the status of the current field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetCurrentFieldValid fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>field_valid_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is past the last field of the message
Description	TipcMsgGetCurrentFieldKnown checks to see if the current field has a known value. A <i>field_valid_return</i> of TRUE indicates that the <i>unknown_value</i> property of the current field is not set and that the value of the field can be retrieved from the message. A <i>field_valid_return</i> of FALSE indicates that the <i>unknown_value</i> property of the current field is set, and that there is no value in the field that can be retrieved.
Caution	If TipcMsgGetCurrentFieldValid returns FALSE, it does not store a value in <i>field_valid_return</i> .
See Also	TipcMsgAppendUnknown, TipcMsgNextUnknown

Examples This example creates a message, appends a field and gets the status of the current field:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL8 real8;
T_BOOL field_valid;

mt=TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if(mt==NULL) {
    return; /* error */
}

msg=TipcMsgCreate(mt);
if (msg==NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg,"speed limit", 65.0)) {
    return; /* error */
}

if (!TipcMsgGetCurrentFieldKnown(msg, &field_valid)) {
    return; /* error */
}
```

TipcMsgGetDeliveryMode

Name TipcMsgGetDeliveryMode — get the delivery mode of a message

Synopsis `T_BOOL TipcMsgGetDeliveryMode(msg, delivery_mode_return)`
`T_IPC_MSG msg;`
`T_IPC_DELIVERY_MODE *delivery_mode_return;`

Arguments *msg* — message to get delivery mode from
delivery_mode_return — storage for message delivery mode

Return Values TRUE if the delivery mode was successfully retrieved from the message, FALSE otherwise.

Diagnostics If TipcMsgGetDeliveryMode fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *msg* was null or *delivery_mode_return* was null
- T_ERR_MSG_INVALID — *msg* was not a valid message

Description TipcMsgGetDeliveryMode gets the delivery mode of a message. The delivery mode of a message controls what level of guarantee is used when the message is sent through a connection with TipcConnMsgSend. When a message is created, its delivery mode is initialized to the message type delivery mode. This allows a default value to easily be set for all messages of a certain type. See TipcMtGetDeliveryMode for more information on message type delivery modes.

The default message delivery mode is T_IPC_DELIVERY_BEST_EFFORT. In this mode, no special actions are taken to ensure delivery of sent messages. The message is delivered unless network failures or process failures occur. If the message is not delivered, there is no way for the sender to know that delivery failed. When there is a failure, it is possible for some messages to be lost or be delivered in a different order than the order in which they were published.

In the case of a network or process failure, a more useful delivery mode is T_IPC_DELIVERY_ORDERED. Messages can still be lost in the event of a failure, but all delivered messages are received in the order in which they were published. Because Java Message Service (JMS) cannot handle messages that come out of order, you must use the T_IPC_DELIVERY_ORDERED delivery mode or one of the GMD delivery modes if you are sending messages to a JMS client.

The difference between the other delivery modes and the GMD delivery modes is that in GMD, a copy of the message is kept when the message is sent and acknowledgements are sent when the message is received. No acknowledgements (ACKs) are used in the ordered or best effort delivery modes, and no copy of the message is kept.

One of the GMD delivery modes is `T_IPC_DELIVERY_SOME`. In this mode, the sending process saves a copy of the message in the connection GMD area until the message is successfully delivered, and the sender can also resend the message if necessary. Delivery is considered successful if the sent message is acknowledged by at least one receiving process.

The other and most robust GMD delivery mode is `T_IPC_DELIVERY_ALL`. In this mode, the sending process saves a copy of the message in the connection GMD area until the message is successfully delivered, and the sender can also resend the message if necessary. Delivery is not considered successful until all receiving processes acknowledge the sent message. For two processes communicating through a non-RTclient and non-RTserver `T_IPC_CONN` connection, `T_IPC_DELIVERY_SOME` and `T_IPC_DELIVERY_ALL` are identical because there is only one process receiving the message. For RTclients, the two modes do differ if more than one RTclient is subscribing to the subject in the destination of the message.

Caution If `TipcMsgGetDeliveryMode` returns `FALSE`, it does not store a value in `delivery_mode_return`.

See Also `TipcMsgSetDeliveryMode`, `TipcMtGetDeliveryMode`, *`TipcDeliveryModeToStr`*

Examples This example prints the delivery mode of a message:

```
T_IPC_MSG msg;
T_IPC_DELIVERY_MODE delivery_mode;
T_STR delivery_mode_str;

if (!TipcMsgGetDeliveryMode(msg, &delivery_mode)) {
    return; /* error */
}

/* convert the delivery mode to a string to make it easier to read */
if (!TipcDeliveryModeToStr(delivery_mode, &delivery_mode_str)) {
    return; /* error */
}

TutOut("The message has delivery mode %s\n", delivery_mode_str);
```

TipcMsgGetDeliveryTimeout

Name	TipcMsgGetDeliveryTimeout — get the delivery timeout of a message in seconds
Synopsis	<pre>T_BOOL TipcMsgGetDeliveryTimeout(msg, delivery_timeout_return) T_IPC_MSG msg; T_REAL8 *delivery_timeout_return;</pre>
Arguments	<i>msg</i> — message to get delivery timeout from <i>delivery_timeout_return</i> — storage for message delivery timeout
Return Values	TRUE if the delivery timeout was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetDeliveryTimeout fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>delivery_timeout_return</i> was null • T_ERR_DOESNT_EXIST — the timeout has not yet been set for this message. This is a benign error. The timeout defaults to the connection’s delivery timeout when the message is sent.
Description	TipcMsgGetDeliveryTimeout gets the delivery timeout of a message in seconds. The delivery timeout of a message controls how long to wait for guaranteed message delivery to complete when a message is sent through a connection with TpcConnMsgSend. When a message is created, its delivery timeout is initialized to the message type delivery timeout, which defaults to UNKNOWN. This allows a default value to be easily set for all messages of a certain type. See TpcMtGetDeliveryTimeout for more information on message type delivery timeouts. When a message is sent using GMD on a connection, the connection delivery timeout is used if the message delivery timeout is not set. The delivery timeout of a message is never sent through the connection, as it is only needed by the sending process.
Caution	None
See Also	TipcMsgSetDeliveryTimeout, TpcMtGetDeliveryTimeout
Examples	This example gets the delivery timeout of a message: <pre>T_REAL8 delivery_timeout; if (!TipcMsgGetDeliveryTimeout(msg, &delivery_timeout)) { return; /* error */ }</pre>

TipcMsgGetDest

Name	TipcMsgGetDest — get the destination property of a message
Synopsis	<pre>T_BOOL TipcMsgGetDest(msg, dest_return) T_IPC_MSG msg; T_STR *dest_return;</pre>
Arguments	<i>msg</i> — message to get destination from <i>dest_return</i> — storage for message destination
Return Values	TRUE if the destination was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetDest fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>dest_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetDest gets the destination property of a message. The destination property is a string whose meaning depends on how the message is being used. Messages sent with publish-subscribe between RTclients by RTserver, for example, use a subject name for the message sender and destination properties. Other applications of messages are free to use the sender and destination for other purposes. Message sender and destination properties can be null if they are not needed.
Caution	If TipcMsgGetDest returns FALSE, it does not store a value in <i>dest_return</i>. Do not modify the value stored in <i>dest_return</i>. It points directly into an internal data structure.
See Also	TipcMsgGetSender, TipcMsgSetDest
Examples	This example prints the destination of a message: <pre>T_STR dest; if (!TipcMsgGetDest(msg, &dest)) { return; /* error */ } /* sender and destination can be NULL, so we have to be careful */ TutOut("msg has destination %s\n", dest ? dest : "<NULL>");</pre>

TipcMsgGetExpiration

Name	TipcMsgGetExpiration — get the expiration property of a message
Synopsis	<pre>T_BOOL TipcMsgGetExpiration(msg, expiration_return) T_IPC_MSG msg; T_REAL8 *expiration_return;</pre>
Arguments	<i>msg</i> — message to get the expiration of <i>expiration_return</i> — storage for message expiration time
Return Values	TRUE if the expiration was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetExpiration fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>expiration_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetExpiration gets the expiration, also known as time to live, property of a message. The expiration property controls how long, in seconds, the message should be in existence. Applications of messages are free to use the expiration for any purpose. A value of 0.0 indicates that the message does not expire.
Caution	None
See Also	TipcMsgSetExpiration
Examples	This example gets the expiration property of a message: <pre>T_REAL8 expiration; if (!TipcMsgGetExpiration(msg, &expiration)) { return; /* error */ }</pre>

TipcMsgGetHeaderStrEncode

Name	TipcMsgGetHeaderStrEncode — get the header string encode property of a message
Synopsis	<pre>T_BOOL TipcMsgGetHeaderStrEncode(msg, header_str_encode_return) T_IPC_MSG msg; T_BOOL *header_str_encode_return;</pre>
Arguments	<i>msg</i> — message to get header string encode property from <i>header_str_encode_return</i> — storage for message header string encode property
Return Values	TRUE if the header string encode property was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetHeaderStrEncode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>header_str_encode_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetHeaderStrEncode gets the header string encode property of a message, which controls whether the message header string properties are converted into four-byte integers when the message is sent through a connection with TipcConnMsgSend. Enabling this property compresses the message header so that less network bandwidth is used, but this compression requires more CPU utilization. When a message is created, its header string encode property is initialized to the message type header string encode property. This allows a default value to easily be set for all messages of a certain type. See TipcMtGetHeaderStrEncode for more information on the message type header string encode property.
Caution	If TipcMsgGetHeaderStrEncode returns FALSE, it does not store a value in <i>header_str_encode_return</i> .
See Also	TipcMsgSetHeaderStrEncode, TipcMtGetHeaderStrEncode
Examples	This example prints the header string encode property of a message: <pre>T_BOOL header_str_encode; if (!TipcMsgGetHeaderStrEncode(msg, &header_str_encode)) { return; /* error */ } TutOut("The message %s encoding header strings.\n", header_str_encode ? "is" : "is not");</pre>

TipcMsgGetLbMode

Name TipcMsgGetLbMode — get the load balancing mode of a message

Synopsis `T_BOOL TipcMsgGetLbMode(msg, lb_mode_return)`
`T_IPC_MSG msg;`
`T_IPC_LB_MODE *lb_mode_return;`

Arguments *msg* — message to get load balancing mode from
lb_mode_return — storage for message load balancing mode

Return Values TRUE if the load balancing mode was successfully retrieved from the message, FALSE otherwise.

Diagnostics If TipcMsgGetLbMode fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *msg* or *lb_mode_return* was null
- T_ERR_MSG_INVALID — *msg* was not a valid message

Description TipcMsgGetLbMode gets the load balancing mode of a message. The load balancing mode of a message controls how (or if) the message is load balanced among RTclients subscribed to a load balanced subject. When a message is created, its load balancing mode is initialized to the message type load balancing mode. This allows a default value to easily be set for all messages of a certain type. See TipcMtGetLbMode for more information on message type load balancing modes.

The default load balancing mode is T_IPC_LB_NONE. In this mode, no load balancing is done. The message is delivered to all RTclients subscribing to the subject to which the message is published.

The other load balancing modes are T_IPC_LB_ROUND_ROBIN, T_IPC_LB_WEIGHTED, and T_IPC_LB_SORTED. In these modes, one and only one RTclient in the list of potential receivers receives the message. In ROUND_ROBIN mode, each successive message is sent to the next RTclient in a circular list of RTclients, to distribute the load evenly. WEIGHTED mode sends the message to the most responsive RTclient (as measured by the fewest pending GMD acknowledgments), reasoning that this is also the RTclient with the least load. In SORTED mode, each message is sent to the RTclient whose unique subject name sorts lowest lexicographically.

For more information on load balancing, see the *TIBCO SmartSockets User’s Guide*.

Caution None

See Also TipcMsgSetLbMode, TipcMtSetLbMode, TipcMtGetLbMode, TipcLbModeToStr

Examples This example retrieves the load balancing mode for a message:

```
T_IPC_MSG msg;
T_IPC_LB_MODE lb_mode;

if (!TipcMsgGetLbMode(msg, &lb_mode)) {
    return; /* error */
}
```

TpcMsgGetMessageId

Name	TpcMsgGetMessageId — get the message identifier property of a message
Synopsis	<pre>T_BOOL TpcMsgGetMessageId(msg, message_id_return) T_IPC_MSG msg; T_STR *message_id_return;</pre>
Arguments	<i>msg</i> — message to get identifier from <i>message_id_return</i> — storage for message identifier
Return Values	TRUE if the identifier was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TpcMsgGetMessageId fails, it returns FALSE and set the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>message_id_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TpcMsgGetMessageId gets the unique message identifier property of a message. The message identifier property is a string that represents a unique identifier for the message. Applications of messages are free to use the message identifier for any purpose.
Caution	If TpcMsgGetMessageId returns FALSE, it does not store a value in <i>message_id_return</i> . Do not modify the value stored in <i>message_id_return</i> . It points directly into an internal data structure.
See Also	TpcMsgGenerateMessageId, TpcMsgSetCorrelationId, TpcMsgGetCorrelationId
Examples	This example prints the message identifier of a message: <pre>T_STR id; if (!TpcMsgGetMessageId(msg, &id)) { return; /* error */ } TutOut("msg had identifier %s\n", id ? id : "<NULL>");</pre>

TipcMsgGetNameCurrent

Name	TipcMsgGetNameCurrent — get the name of the current field
Synopsis	<pre>T_BOOL TipcMsgGetNameCurrent(msg, name_return) T_IPC_MSG msg; T_STR *name_return;</pre>
Arguments	<i>msg</i> — message to get field name from <i>name_return</i> — name of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	None
Description	TipcMsgGetNameCurrent gets the name of the current field (the field that would be returned by a TipcMsgNext* function). If the current field does not have a name, this function fails.
Examples	This example gets the name of the current field: <pre>T_BOOL status; T_STR name; status = TipcMsgGetNameCurrent(msg, &name); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedBinary

Name	TipcMsgGetNamedBinary — get a field containing a BINARY value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedBinary(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_PTR *value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedBinary fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedBinary gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: • No field by this name exists. • The field associated with this name is not a BINARY field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedBinary returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedBinary, TipcMsgUpdateNamedBinary, TipcMsgNextBinary

Examples This example gets a BINARY field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_PTR value;
T_INT4 size;
status = TipcMsgGetNamedBinary(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedBool

Name	TipcMsgGetNamedBool — get a field with a BOOL value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedBool(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_BOOL *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedBool fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedBool gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not a BOOL field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedBool returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedBool, TipcMsgUpdateNamedBool, TipcMsgNextBool
Examples	This example gets a BOOL field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_BOOL value; status = TipcMsgGetNamedBool(msg, "foo", &value); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedBoolArray

Name	TipcMsgGetNamedBoolArray — get a field containing an array of BOOL values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedBoolArray(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_BOOL **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — return pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedBoolArray fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedBoolArray gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: • No field by this name exists. • The field associated with this name is not a BOOL_ARRAY field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedBoolArray returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedBoolArray, TipcMsgUpdateNamedBoolArray, TipcMsgNextBoolArray

Examples This example gets a BOOL_ARRAY field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_BOOL *value;
T_INT4 size;
status = TipcMsgGetNamedBoolArray(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedByte

Name	TipcMsgGetNamedByte — get a field with a BYTE value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedByte(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_IPC_BYTE *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedByte fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedByte gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not a BYTE field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedByte returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedByte, TipcMsgUpdateNamedByte, TipcMsgNextByte
Examples	This example gets a BYTE field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_UCHAR value; status = TipcMsgGetNamedByte(msg, "foo", &value); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedChar

Name	TipcMsgGetNamedChar — get a field with a character value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedChar(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_CHAR *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedChar fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedChar gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not a CHAR field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedChar returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedChar, TipcMsgUpdateNamedChar, TipcMsgNextChar
Examples	This example gets a CHAR field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_CHAR value; status = TipcMsgGetNamedChar(msg, "foo", &value); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedInt2

Name	TipcMsgGetNamedInt2 — get a field with an INT2 value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedInt2(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_INT2 *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedInt2 fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedInt2 gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not an INT2 field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedInt2 returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedInt2, TipcMsgUpdateNamedInt2, TipcMsgNextInt2
Examples	This example gets an INT2 field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_INT2 value; status = TipcMsgGetNamedInt2(msg, "foo", &value); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedInt2Array

Name	TipcMsgGetNamedInt2Array — get a field containing an array of INT2 values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedInt2Array(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_INT2 **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedInt2Array fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedInt2Array gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: • No field by this name exists. • The field associated with this name is not an INT2_ARRAY field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedInt2Array returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedInt2Array, TipcMsgUpdateNamedInt2Array, TipcMsgNextInt2Array

Examples This example gets an INT2_ARRAY field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_INT2 *value;
T_INT4 size;
status = TipcMsgGetNamedInt2Array(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```

TpcMsgGetNamedInt4

Name	TpcMsgGetNamedInt4 — get a field with an INT4 value from a message, using a name
Synopsis	<pre>T_BOOL TpcMsgGetNamedInt4(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_INT4 *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TpcMsgGetNamedInt4 fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TpcMsgGetNamedInt4 gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not an INT4 field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TpcMsgGetNamedInt4 returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TpcMsgAddNamedInt4, TpcMsgUpdateNamedInt4, TpcMsgNextInt4
Examples	This example gets an INT4 field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_INT4 value; status = TpcMsgGetNamedInt4(msg, "foo", &value); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedInt4Array

Name	TipcMsgGetNamedInt4Array — get a field containing an array of INT4 values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedInt4Array(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_INT4 **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedInt4Array fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedInt4Array gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: • No field by this name exists. • The field associated with this name is not an INT4_ARRAY field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedInt4Array returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedInt4Array, TipcMsgUpdateNamedInt4Array, TipcMsgNextInt4Array

Examples This example gets an INT4_ARRAY field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_INT4 *value;
T_INT4 size;
status = TipcMsgGetNamedInt4Array(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedInt8

Name	TipcMsgGetNamedInt8 — get a field with an INT8 value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedInt8(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_INT8 *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — return pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedInt8 fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedInt8 gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not an INT8 field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedInt8 returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedInt8, TipcMsgUpdateNamedInt8, TipcMsgUpdateNamedInt8

Examples This example gets an INT8 field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_INT8 value;

status = TipcMsgGetNamedInt8(msg, "foo", &value);
if (status == FALSE) {
    return; /* error */
}
```


TipcMsgGetNamedInt8Array

Name	TipcMsgGetNamedInt8Array — get a field containing an array of INT8 values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedInt8Array(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_INT8 **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedInt8Array fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedInt8Array gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: • No field by this name exists. • The field associated with this name is not an INT8_ARRAY field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedInt8Array returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedInt8Array, TipcMsgUpdateNamedInt8Array, TipcMsgNextInt8Array

Examples This example gets an INT8_ARRAY field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_INT8 *value;
T_INT4 size;
status = TipcMsgGetNamedInt8Array(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedMsg

Name	TipcMsgGetNamedMsg — get a field with a MSG from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedMsg(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_IPC_MSG *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedMsg fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedMsg gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not a MSG field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedMsg returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedMsg, TipcMsgUpdateNamedMsg, TipcMsgNextMsg
Examples	This example gets a MSG field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_IPC_MSG value; status = TipcMsgGetNamedMsg(msg, "foo", &value); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedMsgArray

Name	TipcMsgGetNamedMsgArray — get a field containing an array of MSG values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedMsgArray(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_IPC_MSG **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — return pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedMsgArray fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedMsgArray gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: • No field by this name exists. • The field associated with this name is not a MSG_ARRAY field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedMsgArray returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedMsgArray, TipcMsgUpdateNamedMsgArray, TipcMsgNextMsgArray

Examples This example gets a MSG_ARRAY field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_IPC_MSG *value;
T_INT4 size;
status = TipcMsgGetNamedMsgArray(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedReal4

Name	TipcMsgGetNamedReal4 — get a field with a REAL4 value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedReal4(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_REAL4 *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedReal4 fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedReal4 gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not a REAL4 field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedReal4 returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedReal4, TipcMsgUpdateNamedReal4, TipcMsgNextReal4
Examples	This example gets a REAL4 field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_REAL4 value; status = TipcMsgGetNamedReal4(msg, "foo", &value); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedReal4Array

Name	TipcMsgGetNamedReal4Array — get a field containing an array of REAL4 values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedReal4Array(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_REAL4 **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedReal4Array fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedReal4Array gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: • No field by this name exists. • The field associated with this name is not a REAL4_ARRAY field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedReal4Array returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedReal4Array, TipcMsgUpdateNamedReal4Array, TipcMsgNextReal4Array

Examples This example gets a REAL4_ARRAY field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_REAL4 *value;
T_INT4 size;
status = TipcMsgGetNamedReal4Array(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```


TipcMsgGetNamedReal8

Name	TipcMsgGetNamedReal8 — get a field with a REAL8 value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedReal8(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_REAL8 *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedReal8 fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedReal8 gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not a REAL8 field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedReal8 returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedReal8, TipcMsgUpdateNamedReal8, TipcMsgNextReal8
Examples	This example gets a REAL8 field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_REAL8 value; status = TipcMsgGetNamedReal8(msg, "foo", &value); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedReal8Array

Name	TipcMsgGetNamedReal8Array — get a field containing an array of REAL8 values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedReal8Array(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_REAL8 **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedReal8Array fails, it returns FALSE and sets the global SmartSockets error number to: - T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null - T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedReal8Array gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: - No field by this name exists. - The field associated with this name is not a REAL8_ARRAY field. - The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedReal8Array returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedReal8Array, TipcMsgUpdateNamedReal8Array, TipcMsgNextReal8Array

Examples This example gets a REAL8_ARRAY field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_REAL8 *value;
T_INT4 size;
status = TipcMsgGetNamedReal8Array(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedReal16

Name	TipcMsgGetNamedReal16 — get a field with a REAL16 value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedReal16(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_REAL16 *value_return;</pre>
Arguments	<i>msg</i> — message to append field onto <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedReal16 fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedReal16 gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not a REAL16 field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedReal16 returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
	Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.
See Also	TipcMsgAddNamedReal16, TipcMsgUpdateNamedReal16, TipcMsgNextReal16

Examples This example gets a REAL16 field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_REAL16 value;

status = TipcMsgGetNamedReal16(msg, "foo", &value);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedReal16Array

Name	TipcMsgGetNamedReal16Array — get a field containing an array of REAL16 values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedReal16Array(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_REAL16 **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedReal16Array fails, it returns FALSE and sets the global SmartSockets error number to: - T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null - T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedReal16Array gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: - No field by this name exists. - The field associated with this name is not a REAL16_ARRAY field. - The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedReal16Array returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
 Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.	

See Also `TipcMsgAddNamedReal16Array`, `TipcMsgUpdateNamedReal16Array`,
`TipcMsgNextReal16Array`

Examples This example gets a REAL16_ARRAY field from an existing message:

```
T_IPC_MSG msg;  
T_BOOL status;  
T_REAL16 *value;  
T_INT4 size;  
status = TipcMsgGetNamedReal16Array(msg, "foo", &value, &size);  
if (status == FALSE) {  
    return; /* error */  
}
```

TipcMsgGetNamedStr

Name	TipcMsgGetNamedStr — get a field with a STR value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedStr(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_STR *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — return pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedStr fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedStr gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not a STR field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedStr returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedStr, TipcMsgUpdateNamedStr, TipcMsgNextStr
Examples	This example gets a STR field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_STR value; status = TipcMsgGetNamedStr(msg, "foo", &value); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedStrArray

Name	TipcMsgGetNamedStrArray — get a field containing an array of STR values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedStrArray(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_STR **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedStrArray fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedStrArray gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: • No field by this name exists. • The field associated with this name is not a STR_ARRAY field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedStrArray returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedStrArray, TipcMsgUpdateNamedStrArray, TipcMsgNextStrArray

Examples This example gets a STR_ARRAY field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_STR *value;
T_INT4 size;
status = TipcMsgGetNamedStrArray(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedTimestamp

Name	TipcMsgGetNamedTimestamp — get a field with a <code>TIMESTAMP</code> value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedTimestamp(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_REAL8 *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedTimestamp fails, it returns FALSE and sets the global SmartSockets error number to: • <code>T_ERR_NULL_PTR</code> — <i>msg</i> was null or <i>name</i> was null • <code>T_ERR_MSG_INVALID</code> — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedTimestamp gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not a <code>TIMESTAMP</code> field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedTimestamp returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedTimestamp, TipcMsgUpdateNamedTimestamp, TipcMsgNextTimestamp

Examples This example gets a TIMESTAMP field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_REAL8 value;

status = TipcMsgGetNamedTimestamp(msg, "foo", &value);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedTimestampArray

Name	TipcMsgGetNamedTimestampArray — get a field containing an array of TIMESTAMP values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedTimestampArray(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_REAL8 **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedTimestampArray fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedTimestampArray gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: • No field by this name exists. • The field associated with this name is not a TIMESTAMP_ARRAY field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedTimestampArray returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedTimestampArray, TipcMsgUpdateNamedTimestampArray, TipcMsgNextReal4Array

Examples This example gets a `TIMESTAMP_ARRAY` field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_REAL8 *value;
T_INT4 size;
status = TipcMsgGetNamedTimestampArray(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedUnknown

Name	TipcMsgGetNamedUnknown — get a field with an unknown or non-existent value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedUnknown(msg, name) T_IPC_MSG msg; T_STR name;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedUnknown fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedUnknown gets the field specified by the name. If no field by this name exists, or the field associated with this name is not unknown, this function fails.
Caution	None
See Also	TipcMsgAppendUnknown, TipcMsgUpdateNamedUnknown, TipcMsgNextUnknown
Examples	This example gets a named field of unknown type from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; status = TipcMsgGetNamedUnknown(msg, "foo"); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedUtf8

Name	TipcMsgGetNamedUtf8 — get a field with a UTF8 value from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedUtf8(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_STR *value_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedUtf8 fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedUtf8 gets the field specified by the name. The function fails if: • No field by this name exists. • The field associated with this name is not a UTF8 field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedUtf8 returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedUtf8, TipcMsgUpdateNamedUtf8, TipcMsgNextUtf8
Examples	This example gets a UTF8 field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_STR value; status = TipcMsgGetNamedUtf8(msg, "foo", &value); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNamedUtf8Array

Name	TipcMsgGetNamedUtf8Array — get a field containing an array of UTF8 values from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetNamedUtf8Array(msg, name, value_return, size_return) T_IPC_MSG msg; T_STR name; T_STR **value_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>value_return</i> — return pointer to value of the field <i>size_return</i> — pointer to size of the field, in bytes
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgGetNamedUtf8Array fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>name</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNamedUtf8Array gets the field specified by the name. The returned pointer should be considered read-only. The function fails if: • No field by this name exists. • The field associated with this name is not a UTF8_ARRAY field. • The field associated with this name is unknown. See the <i>TIBCO SmartSockets User's Guide</i> for more information on unknown field values.
Caution	If TipcMsgGetNamedUtf8Array returns FALSE, it does not store a value in <i>value_return</i>. Do not modify the value in <i>value_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedUtf8Array, TipcMsgUpdateNamedUtf8Array, TipcMsgNextUtf8Array

Examples This example gets a UTF8_ARRAY field from an existing message:

```
T_IPC_MSG msg;
T_BOOL status;
T_STR *value;
T_INT4 size;
status = TipcMsgGetNamedUtf8Array(msg, "foo", &value, &size);
if (status == FALSE) {
    return; /* error */
}
```

TipcMsgGetNamedXml

Name	TipcMsgGetNamedXml — Get a named XML field from a message
Synopsis	<pre>T_BOOL TipcMsgGetNamedXml(msg, name, value_return) T_IPC_MSG msg; T_STR name; T_XML *xml_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>name</i> — name of the field <i>xml_return</i> — return pointer to value of the field
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	If TipcMsgNextXml fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>xml_return</i> was null. • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message. • T_ERR_MSG_EOM — the current field is at the end of message. • T_ERR_FT_MISMATCH — the current field is of the wrong type. • T_ERR_VAL_UNKNOWN — the current field is of unknown value.
Description	TipcMsgGetNamedXml gets the field specified by the name. If no field by this name exists, or the field associated with this name is unknown, this function fails.
Caution	If TipcMsgGetNamedXml returns FALSE, it does not store a value in <i>xml_return</i>. Do not modify the value stored in <i>xml_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAddNamedXml, TipcMsgUpdateNamedXml, TipcMsgNextXml
Examples	<pre>T_IPC_MSG msg; T_BOOL status; T_XML xml; status = TipcMsgGetNamedXml(msg, "foo", &xml); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetNumFields

Name	TipcMsgGetNumFields — get the number of fields in a message
Synopsis	<pre>T_BOOL TipcMsgGetNumFields(msg, num_fields_return) T_IPC_MSG msg; T_INT4 *num_fields_return;</pre>
Arguments	<i>msg</i> — message to get number of fields from <i>num_fields_return</i> — storage for number of fields
Return Values	TRUE if the number of fields was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetNumFields fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>num_fields_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetNumFields gets the number of fields in a message. While it is possible to calculate the number of fields by using TipcMsgTraverse, TipcMsgGetNumFields is much faster.
Caution	If TipcMsgGetNumFields returns FALSE, it does not store a value in <i>num_fields_return</i> .
See Also	TipcMsgTraverse
Examples	This example prints the number of fields in a message: <pre>void print_num_fields(msg) T_IPC_MSG msg; { T_INT4 num_fields; if (!TipcMsgGetNumFields(msg, &num_fields)) { return; /* error */ } TutOut("The message has %d fields.\n", num_fields); } /* print_num_fields */</pre>

TipcMsgGetPacketSize

Name	TipcMsgGetPacketSize — get the packet size of a message in bytes
Synopsis	<pre>T_BOOL TipcMsgGetPacketSize(msg, packet_size_return) T_IPC_MSG msg; T_UINT4 *packet_size_return;</pre>
Arguments	<i>msg</i> — message to get packet size from <i>packet_size_return</i> — storage for message packet size
Return Values	TRUE if the packet size was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetPacketSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>packet_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetPacketSize gets the packet size of a message. A message is implemented internally with several blocks of memory: • one block for holding the fixed-size properties (such as priority and reference count) and pointers to the variable-length properties (such as the destination and data properties) • several blocks for the variable-length properties TipcMsgGetPacketSize calculates how many bytes of memory are needed for the entire message. TipcMsgGetPacketSize is useful for determining the size of a message for bandwidth estimates.
Caution	If TipcMsgGetPacketSize returns FALSE, it does not store a value in <i>packet_size_return</i> .
See Also	None
Examples	This example determines the size of a message: <pre>T_UINT4 packet_size; /* get the number of bytes needed for the single packet of memory */ if (!TipcMsgGetPacketSize(msg, &packet_size)) { return; /* error */ }</pre>

TpcMsgGetPriority

Name TpcMsgGetPriority — get the priority of a message

Synopsis `T_BOOL TpcMsgGetPriority(msg, priority_return)`
`T_IPC_MSG msg;`
`T_INT2 *priority_return;`

Arguments *msg* — message to get priority from
priority_return — storage for message priority

Return Values TRUE if the priority was successfully retrieved from the message, FALSE otherwise.

Diagnostics If TpcMsgGetPriority fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *msg* was null or *priority_return* was null
- T_ERR_MSG_INVALID — *msg* was not a valid message

Description TpcMsgGetPriority gets the priority of a message. Message priorities are used by TpcConnRead to queue incoming messages in priority order. When a message is created, its priority is initialized to the message type priority (if set) or to the value of the option Default_Msg_Priority (if the message type priority is unknown). This allows a default value to be set easily for all messages of a certain type or for all messages.

Caution If TpcMsgGetPriority returns FALSE, it does not store a value in *priority_return*.

See Also TpcConnRead, TpcMsgSetPriority, TpcMtGetPriority

Examples This example prints the priority of a message:

```
T_INT2 priority;

if (!TpcMsgGetPriority(msg, &priority)) {
    return; /* error */
}

TutOut("msg has priority %d\n", priority);
```

TipcMsgGetReadOnly

Name	TipcMsgGetReadOnly — get read-only property of message
Synopsis	<pre>T_BOOL TipcMsgGetReadOnly(msg, read_only_return) T_IPC_MSG msg; T_BOOL *read_only_return;</pre>
Arguments	<i>msg</i> — message to get read-only property from <i>read_only_return</i> — storage for read-only property
Return Values	TRUE if the read-only property was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetReadOnly fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>read_only_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetReadOnly gets the read-only property of a message. A message is marked read only if the message is embedded as a field in another message.
Caution	If TipcMsgGetReadOnly returns FALSE, it does not store a value in <i>read_only_return</i> .
See Also	TipcMsgClone
Examples	This example prints the read-only property of a message: <pre>void print_read_only(msg) T_IPC_MSG msg; { T_BOOL read_only; if (!TipcMsgGetReadOnly(msg, &read_only)) { return; /* error */ } TutOut("The message is %smarked read-only.\n", read_only ? "" : "not "); } /* print_read_only */</pre>

TipcMsgGetRefCount

Name	TipcMsgGetRefCount — get the reference count of a message
Synopsis	<pre>T_BOOL TipcMsgGetRefCount(msg, ref_count_return) T_IPC_MSG msg; T_INT2 *ref_count_return;</pre>
Arguments	<i>msg</i> — message to get reference count from <i>ref_count_return</i> — storage for message reference count
Return Values	TRUE if the reference count was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetRefCount fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>ref_count_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetRefCount gets the reference count of a message. The reference count property can be used to prevent a message from being destroyed. See TipcMsgIncrRefCount for more information on message reference counts.
Caution	If TipcMsgGetRefCount returns FALSE, it does not store a value in <i>ref_count_return</i> .
See Also	TipcMsgDestroy, TipcMsgIncrRefCount
Examples	This example prints the reference count of a message: <pre>T_IPC_MSG msg; T_INT2 ref_count; if (!TipcMsgGetRefCount(msg, &ref_count)) { return; /* error */ } TutOut("msg has ref count %d\n", ref_count);</pre>

TipcMsgGetReplyTo

Name	TipcMsgGetReplyTo — get the reply to destination property of a message
Synopsis	<pre>T_BOOL TipcMsgGetReplyTo(msg, reply_to_dest_return) T_IPC_MSG msg; T_STR *reply_to_dest_return;</pre>
Arguments	<i>msg</i> — message to get reply to destination from <i>reply_to_dest_return</i> — storage for message reply to destination
Return Values	TRUE if the reply to destination was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetReplyTo fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>reply_to_dest_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetReplyTo gets the reply to destination property of a message. The reply to destination property is a string representing a subject where a reply to the message should be sent.
Caution	If TipcMsgGetReplyTo returns FALSE, it does not store a value in <i>reply_to_dest_return</i>. Do not modify the value stored in <i>reply_to_dest_return</i>. It points directly into an internal data structure.
See Also	TipcMsgSetReplyTo
Examples	This example prints the reply to destination of a message: <pre>T_STR reply_to_dest; if (!TipcMsgGetReplyTo(msg, &reply_to_dest)) { return; /* error */ } TutOut("msg had reply to %s\n", reply_to_dest ? reply_to_dest : "<NULL>");</pre>

TipcMsgGetSender

Name	TipcMsgGetSender — get the sender property of a message
Synopsis	<pre>T_BOOL TipcMsgGetSender(msg, sender_return) T_IPC_MSG msg; T_STR *sender_return;</pre>
Arguments	<i>msg</i> — message to get sender from <i>sender_return</i> — storage for message sender
Return Values	TRUE if the sender was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetSender fails, it returns FALSE and sets the global SmartSockets error number to one of: - T_ERR_NULL_PTR — <i>msg</i> was null or <i>sender_return</i> was null - T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetSender gets the sender property of a message. The sender property is a string whose meaning depends on how the message is being used. Messages sent with publish-subscribe between RTclients by RTserver, for example, use a subject name for the message sender and destination properties. Other applications of messages are free to use the sender and destination for other purposes. Message sender and destination properties can be null if they are not needed.
Caution	If TipcMsgGetSender returns FALSE, it does not store a value in <i>sender_return</i>. Do not modify the value stored in <i>sender_return</i>. It points directly into an internal data structure.
See Also	TipcMsgGetDest, TipcMsgSetType
Examples	This example prints the sender of a message: <pre>T_STR sender; if (!TipcMsgGetSender(msg, &sender)) { return; /* error */ } /* sender and destination can be NULL, so we have to be careful */ TutOut("msg has sender %s\n", sender ? sender : "<NULL>");</pre>

TipcMsgGetSenderTimestamp

Name	TipcMsgGetSenderTimestamp — get the sender timestamp property of a message
Synopsis	<pre>T_BOOL TipcMsgGetSenderTimestamp(msg, sender_timestamp_return) T_IPC_MSG msg; T_REAL8 *sender_timestamp_return;</pre>
Arguments	<i>msg</i> — message to get sender timestamp from <i>sender_timestamp_return</i> — storage for message sender timestamp
Return Values	TRUE if the sender timestamp was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetSenderTimestamp fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>sender_timestamp_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetSenderTimestamp gets the sender timestamp property of a message. A value of 0.0 indicates that the sender timestamp property was not set.
Caution	Do not modify the value stored in <i>sender_timestamp_return</i> . It points directly into an internal data structure.
See Also	TipcMsgSetSenderTimestamp
Examples	This example gets the sender timestamp of a message: <pre>T_REAL8 sender_timestamp; if (!TipcMsgGetSenderTimestamp(msg, &sender_timestamp)) { return; /* error */ }</pre>

TipcMsgGetSeqNum

Name	TipcMsgGetSeqNum — get the sequence number of a message
Synopsis	<pre>T_BOOL TipcMsgGetSeqNum(msg, seq_num_return) T_IPC_MSG msg; T_INT4 *seq_num_return;</pre>
Arguments	<i>msg</i> — message to get sequence number from <i>seq_num_return</i> — storage for sequence number
Return Values	TRUE if the sequence number was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetSeqNum fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>seq_num_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetSeqNum gets the sequence number of a message. The sequence number is used by GMD to detect duplicate messages. When a message with a delivery mode of T_IPC_DELIVERY_SOME or T_IPC_DELIVERY_ALL is sent through a connection, TipcConnMsgSend increments an internal per-connection sequence number and sets the sequence number of the message to the incremented value. If the message is later resent because of a GMD failure, TipcConnGmdMsgResend preserves the sequence number, which allows the receiving processes to detect the duplicate sequence number and thus avoid processing the same message twice.
Caution	If TipcMsgGetSeqNum returns FALSE, it does not store a value in <i>seq_num_return</i> .
See Also	TipcConnMsgSend, TipcConnGmdMsgResend
Examples	This example prints the sequence number of a message: <pre>void T_ENTRY print_seq_num(msg) T_IPC_MSG msg; { T_INT4 seq_num; if (!TipcMsgGetSeqNum(msg, &seq_num)) { return; /* error */ } TutOut("The message sequence number is %d.\n", seq_num); } /* print_seq_num */</pre>

TipcMsgGetType

Name	TipcMsgGetType — get the type of a message
Synopsis	<pre>T_BOOL TipcMsgGetType(msg, mt_return) T_IPC_MSG msg; T_IPC_MT *mt_return;</pre>
Arguments	<i>msg</i> — message to get type from <i>mt_return</i> — storage for message type
Return Values	TRUE if the type was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetType fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>mt_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_DOESNT_EXIST — <i>msg</i> does not have a valid type
Description	TipcMsgGetType gets the type of a message. The type identifies the category of message being manipulated. Most connection callback types use a message type, for example, to allow various kinds of execution based on specific message types.
Caution	If TipcMsgGetType returns FALSE, it does not store a value in <i>mt_return</i> .
See Also	TipcMsgSetType, TipcMtLookupByNum, TipcMsgGetTypeNum

Examples This gets the type of a message and prints the name and number of the message type:

```
void T_ENTRY print_msg_type_name(msg)
T_IPC_MSG msg;
{
    T_IPC_MT mt;
    T_STR name;
    T_INT4 num;
    if (!TipcMsgGetType(msg, &mt)) {
        return; /* error */
    }
    if (!TipcMtGetName(mt, &name)) {
        return; /* error */
    }
    if (!TipcMtGetNum(mt, &num)) {
        return; /* error */
    }
    TutOut("msg has type %s (number %d)\n", name, num);
} /* print_msg_type_name */
```

TipcMsgGetTypeNamed

Name	TipcMsgGetTypeNamed — get the type of a field from a message, using a name
Synopsis	<pre>T_BOOL TipcMsgGetTypeNamed(msg, name, type) T_IPC_MSG msg; T_STR name; T_IPC_FT *type;</pre>
Arguments	<i>msg</i> — message to get field type from <i>name</i> — name of the field <i>type</i> — return pointer to type of the field, such as T_IPC_FT_INT2 or T_IPC_FT_INT4
Return Values	TRUE if the get was successful, FALSE otherwise.
Diagnostics	None
Description	TipcMsgGetTypeNamed gets the type of the field specified by the name. The type of the field is returned in the type pointer. TipcMsgGetTypeNamed fails if no field by this name exists.
See Also	TipcMsgNextType
Examples	This example gets the type of a field from an existing message: <pre>T_IPC_MSG msg; T_BOOL status; T_IPC_FT type; status = TipcMsgGetTypeNamed(msg, "foo", &type); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgGetTypeNum

Name TipcMsgGetTypeNum — get the message type of a message as an integer

Synopsis `T_BOOL TipcMsgGetTypeNum (msg, type_num_return);`
`T_IPC_MSG msg;`
`T_INT4 *type_num_return;`

Arguments *msg* — message to get the message type of
type_num_return — storage for the message type

Return Values TRUE if the message type was successfully retrieved, FALSE otherwise.

Diagnostics If TipcMsgGetTypeNum fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *msg* was null or *type_num_return* was null
- T_ERR_MSG_INVALID — *msg* was not a valid message

Description TipcMsgGetTypeNum gets the value of the Number property of the message type. Each message type is uniquely identified by the Number property, which is stored internally as an integer. See the *TIBCO SmartSockets User’s Guide* for more information on message types.

Caution None

See Also TipcMsgGetType, TipcMtLookupByNum

Examples This example:

```
T_BOOL status;
T_INT4 mt_num;
T_IPC_MT mt;
T_IPC_MSG msg = data->msg;

status = TipcMsgGetTypeNum(msg, &mt_num);
T_ASSERT(T_TRUE == status);
mt = TipcMtLookupByNum(mt_num);
```


TipcMsgGetUserProp

Name	TipcMsgGetUserProp — get the user-defined property of a message
Synopsis	<pre>T_BOOL TipcMsgGetUserProp(msg, user_prop_return) T_IPC_MSG msg; T_INT4 *user_prop_return;</pre>
Arguments	<i>msg</i> — message to get user-defined property from <i>user_prop_return</i> — storage for message user-defined property
Return Values	TRUE if the user-defined property was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgGetUserProp fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>user_prop_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgGetUserProp gets the user-defined property of a message. You can use the user-defined property for any purpose, such as attaching a version number to messages. This property is not used internally by SmartSockets. When a message is sent through a connection with TipcConnMsgSend, the user-defined property is byte-swapped along with all the other integer message properties. On architectures where a pointer is four bytes, a pointer can be stored in the user-defined property of a message. This is useful for attaching extra temporary data to the message. Use pointers with caution; when a message is sent through a connection, the data the pointer refers to does not exist in the receiving process. In this situation, add extra fields to the message instead of using a pointer in the user-defined property. When a message is created, its user-defined property is initialized to the message type user-defined property. This allows a default value to easily be set for all messages of a certain type.
Caution	If TipcMsgGetUserProp returns FALSE, it does not store a value in <i>user_prop_return</i> .
See Also	TipcMsgSetUserProp, TipcMtGetUserProp

Examples This example prints the user-defined property of a message:

```
T_IPC_MSG msg;
T_INT4 user_prop;

if (!TipcMsgGetUserProp(msg, &user_prop)) {
    return; /* error */
}

TutOut("msg has user prop %d\n", user_prop);
```

TipcMsgIncrRefCount

Name	TipcMsgIncrRefCount — increment the reference count of a message
Synopsis	<pre>T_BOOL TipcMsgIncrRefCount(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to increment reference count for
Return Values	TRUE if the reference count was successfully incremented for the message, FALSE otherwise.
Diagnostics	If TipcMsgIncrRefCount fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgIncrRefCount increments the reference count of a message. The reference count property can be used to prevent a message from being destroyed. This is useful when using functions that always call TipcMsgDestroy, such as TipcConnMainLoop or TipcSrvMainLoop. For example, connection message-related callbacks are only given temporary access to a message, but sometimes these callbacks need to save the messages for later processing. When a message is created or read from a connection, its reference count is initialized to 1. Each call to TipcMsgIncrRefCount increments the reference count. Each call to TipcMsgDestroy decrements the reference count. Once the reference count reaches 0, TipcMsgDestroy permanently destroys the message. While TipcMsgClone can also be used to copy a message, if the message is not going to be modified then TipcMsgIncrRefCount can be used instead. TipcMsgIncrRefCount is much faster than TipcMsgClone because no memory allocation and copying is performed.
Caution	For each call to TipcMsgIncrRefCount there should be one call to TipcMsgDestroy to ensure proper destruction of a message when it is no longer needed, unless the message is passed to another function which takes responsibility for destroying the message (such as TipcSrvMsgInsert).
See Also	TipcMsgDestroy, TipcMsgGetRefCount

Examples This example shows a connection process callback function which increments the reference count of a message and saves the messages for later processing:

```
void T_ENTRY example_process_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG;
{

    do_some_processing_now_on_msg(data->msg);

    if (!TipcMsgIncrRefCount(data->msg)) {
        return; /* error */
    }

    save_msg_for_later_processing(data->msg);
} /* example_process_cb */
```

Once the later processing has been performed, the message can be destroyed by calling TipcMsgDestroy once for each call to TipcMsgIncrRefCount.

TipcMsgNextBinary - TipcMsgTraverse

This chapter covers these TipcMsg API functions:

- *TipcMsgNextBinary*
- *TipcMsgNextBool*
- *TipcMsgNextBoolArray*
- *TipcMsgNextByte*
- *TipcMsgNextChar*
- *TipcMsgNextInt2*
- *TipcMsgNextInt2Array*
- *TipcMsgNextInt4*
- *TipcMsgNextInt4Array*
- *TipcMsgNextInt8*
- *TipcMsgNextInt8Array*
- *TipcMsgNextMsg*
- *TipcMsgNextMsgArray*
- *TipcMsgNextReal4*
- *TipcMsgNextReal4Array*
- *TipcMsgNextReal8*
- *TipcMsgNextReal8Array*
- *TipcMsgNextReal16*
- *TipcMsgNextReal16Array*
- *TipcMsgNextStr*
- *TipcMsgNextStrArray*
- *TipcMsgNextStrReal8*
- *TipcMsgNextTimestamp*
- *TipcMsgNextTimestampArray*
- *TipcMsgNextType*

- *TpcMsgNextUnknown*
- *TpcMsgNextUtf8*
- *TpcMsgNextUtf8Array*
- *TpcMsgNextXml*
- *TpcMsgPrint*
- *TpcMsgPrintError*
- *TpcMsgRead*
- *TpcMsgReadVa*
- *TpcMsgSetArrivalTimestamp*
- *TpcMsgSetCompression*
- *TpcMsgSetCorrelationId*
- *TpcMsgSetCurrent*
- *TpcMsgSetDeliveryMode*
- *TpcMsgSetDeliveryTimeout*
- *TpcMsgSetDest*
- *TpcMsgSetExpiration*
- *TpcMsgSetHeaderStrEncode*
- *TpcMsgSetLbMode*
- *TpcMsgSetNameCurrent*
- *TpcMsgSetNumFields*
- *TpcMsgSetPriority*
- *TpcMsgSetReplyTo*
- *TpcMsgSetSender*
- *TpcMsgSetSenderTimestamp*
- *TpcMsgSetType*
- *TpcMsgSetUserProp*
- *TpcMsgTraverse*

TipcMsgNextBinary

Name	TipcMsgNextBinary — get a BINARY field from a message
Synopsis	<pre>T_BOOL TipcMsgNextBinary(msg, binary_return, size_return) T_IPC_MSG msg; T_PTR *binary_return; T_INT4 *size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>binary_return</i> — storage for BINARY value from field <i>size_return</i> — storage for number of bytes in BINARY field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextBinary fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>binary_return</i> was null, or <i>size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextBinary retrieves the value of the current field of a message, which must be a BINARY (arbitrary binary data) field. If TipcMsgNextBinary succeeds, it advances the current field to the next field in the message.
Caution	If TipcMsgNextBinary returns FALSE, it does not store values in <i>binary_return</i> and <i>size_return</i>. Do not modify the value stored in <i>binary_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendBinary, TipcMsgAppendBinaryPtr

Examples This example reads a BINARY image field from a message and prints the hex representation of the binary data:

```

/* pointers to arrays are a little tricky in C */
char (*image)[1024]; /* fixed size image */
T_INT4 image_size;
T_INT4 i;
T_INT4 j;

if (!TipcMsgNextBinary(msg, &image, &image_size)) {
    return; /* error */
}

/* image_size had better be 1024x1024! */
TutOut("image_size = %d\n", image_size);

for (i = 0; i < 1024; i++) {
    for (j = 0; j < 1024; j++) {
        TutOut("image[%d][%d] = %x\n", i, j, image[i][j]);
    }
}

```


TipcMsgNextBool

Name	TipcMsgNextBool — get a BOOL field from a message
Synopsis	<pre>T_BOOL TipcMsgNextBool(msg, bool_return) T_IPC_MSG msg; T_BOOL *bool_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>bool_return</i> — storage for BOOL value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextBool fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>bool_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextBool retrieves the value of the current field of a message, which must be a BOOL field. If TipcMsgNextBool succeeds, it advances the current field to the next field in the message. If necessary, TipcMsgNextBool converts the field into the data format used by this process.
Caution	If TipcMsgNextBool returns FALSE, it does not store a value in <i>bool_return</i>. Do not modify the value stored in <i>bool_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendBool

Examples This example creates a message, appends a BOOL field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_BOOL value;

#define USER_MT_BOOL_TEST 100
mt = TipcMtCreate("bool_test", USER_MT_BOOL_TEST, "bool");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}
if (!TipcMsgAppendBool(msg, T_TRUE)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextBool(msg, &value)) {
    return; /* error */
}
TutOut("boolean value == %s\n", (value)?"True":"False");
```

TipcMsgNextBoolArray

Name	TipcMsgNextBoolArray — get a field containing an array of BOOL fields from a message
Synopsis	<pre>T_BOOL TipcMsgNextBoolArray(msg, bool_array_return, array_size_return) T_IPC_MSG msg; T_BOOL **bool_array_return; T_INT4 *array_size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>bool_array_return</i> — storage for BOOL_ARRAY value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextBoolArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>bool_array_return</i> was null, or <i>array_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextBoolArray retrieves the value of the current field of a message, which must be a BOOL_ARRAY field. If TipcMsgNextBoolArray succeeds, it advances the current field to the next field in the message. TipcMsgNextBoolArray converts the field into the data format used by this process, if necessary.
Caution	If TipcMsgNextBoolArray returns FALSE, it does not store values in <i>bool_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>bool_array_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendBoolArray, TipcMsgAppendBoolArrayPtr

Examples This example reads a BOOL_ARRAY field from a message and prints the values in the array:

```
T_BOOL *bool_array;
T_BOOL array_size;
T_INT4 counter;

if (!TipcMsgNextBoolArray(msg, &bool_array, &array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    TutOut("bool_array[%d] = " T_BOOL_SPEC "\n", counter,
          bool_array[counter]);
}
```

TipcMsgNextByte

Name	TipcMsgNextByte — get a BYTE field from a message
Synopsis	<pre>T_BOOL TipcMsgNextByte(msg, byte_return) T_IPC_MSG msg; T_UCHAR *byte_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>byte_return</i> — storage for BYTE value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextByte fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>byte_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextByte retrieves the value of the current field of a message, which must be a BYTE field. If TipcMsgNextByte succeeds, it advances the current field to the next field in the message. TipcMsgNextByte converts the field into the data format used by this process, if necessary.
Caution	If TipcMsgNextByte returns FALSE, it does not store a value in <i>byte_return</i>. Do not modify the value stored in <i>byte_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendByte

Examples This example creates a message, appends a BYTE field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_UCHAR value;

#define USER_MT_BYTE_TEST 100
mt = TipcMtCreate("byte_test", USER_MT_BYTE_TEST, "byte");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}
if (!TipcMsgAppendByte(msg, 'c')) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextByte(msg, &value)) {
    return; /* error */
}
TutOut("byte value == %c\n", value);
```

TipcMsgNextChar

Name	TipcMsgNextChar — get a CHAR field from a message
Synopsis	<pre>T_BOOL TipcMsgNextChar(msg, char_return) T_IPC_MSG msg; T_CHAR *char_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>char_return</i> — storage for CHAR value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextChar fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, or <i>char_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextChar retrieves the value of the current field of a message, which must be a CHAR (character) field. If TipcMsgNextChar succeeds, it advances the current field to the next field in the message. TipcMsgNextChar converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextChar returns FALSE, it does not store a value in <i>char_return</i> .
See Also	TipcMsgAppendChar

Examples This example creates a message, appends a CHAR field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_CHAR char_val;

#define USER_MT_CHAR_TEST 100
mt = TipcMtCreate("char_test", USER_MT_CHAR_TEST, "char");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendChar(msg, 'x')) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextChar(msg, &char_val)) {
    return; /* error */
}
TutOut("char_val == %d\n", char_val);
```


TipcMsgNextInt2

Name	TipcMsgNextInt2 — get an INT2 field from a message
Synopsis	<pre>T_BOOL TipcMsgNextInt2(msg, int2_return) T_IPC_MSG msg; T_INT2 *int2_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>int2_return</i> — storage for INT2 value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextInt2 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>int2_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextInt2 retrieves the value of the current field of a message, which must be an INT2 (two-byte integer) field. If TipcMsgNextInt2 succeeds, it advances the current field to the next field in the message. TipcMsgNextInt2 converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextInt2 returns FALSE, it does not store a value in <i>int2_return</i> .
See Also	TipcMsgAppendInt2

Examples This example creates a message, appends an INT2 field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT2 int2_val;

#define USER_MT_INT2_TEST 100
mt = TipcMtCreate("int2_test", USER_MT_INT2_TEST, "int2");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendInt2(msg, 42)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextInt2(msg, &int2_val)) {
    return; /* error */
}
TutOut("int2_val == %d\n", int2_val);
```

TipcMsgNextInt2Array

Name	TipcMsgNextInt2Array — get an INT2_ARRAY field from a message
Synopsis	<pre>T_BOOL TipcMsgNextInt2Array(msg, int2_array_return, array_size_return) T_IPC_MSG msg; T_INT2 **int2_array_return; T_INT4 *array_size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>int2_array_return</i> — storage for INT_ARRAY value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextInt2Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>int2_array_return</i> was null, or <i>array_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextInt2Array retrieves the value of the current field of a message, which must be an INT2_ARRAY (array of two-byte integers) field. If TipcMsgNextInt2Array succeeds, it advances the current field to the next field in the message. TipcMsgNextInt2Array converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextInt2Array returns FALSE, it does not store values in <i>int2_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>int2_array_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendInt2Array, TipcMsgAppendInt2ArrayPtr

Examples This example reads an INT2_ARRAY field from a message and prints the values in the array:

```
T_INT2 *int2_array;
T_INT4 array_size;
T_INT4 counter;

if (!TipcMsgNextInt2Array(msg, &int2_array, &array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    TutOut("int2_array[%d] = %d\n", counter, int2_array[counter]);
}
```

TipcMsgNextInt4

Name	TipcMsgNextInt4 — get an INT4 field from a message
Synopsis	<pre>T_BOOL TipcMsgNextInt4(msg, int4_return) T_IPC_MSG msg; T_INT4 *int4_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>int4_return</i> — storage for INT4 value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextInt4 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>int4_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextInt4 retrieves the value of the current field of a message, which must be an INT4 (four-byte integer) field. If TipcMsgNextInt4 succeeds, it advances the current field to the next field in the message. TipcMsgNextInt4 converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextInt4 returns FALSE, it does not store a value in <i>int4_return</i> .
See Also	TipcMsgAppendInt4

Examples This example creates a message, appends an INT4 field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT4 int4_val;

#define USER_MT_INT4_TEST 100
mt = TipcMtCreate("int4_test", USER_MT_INT4_TEST, "int4");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendInt4(msg, 42)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextInt4(msg, &int4_val)) {
    return; /* error */
}
TutOut("int4_val == %d\n", int4_val);
```

TipcMsgNextInt4Array

Name	TipcMsgNextInt4Array — get a field containing an array of INT4 fields from a message
Synopsis	<pre>T_BOOL TipcMsgNextInt4Array(msg, int4_array_return, array_size_return) T_IPC_MSG msg; T_INT4 **int4_array_return; T_INT4 *array_size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>int4_array_return</i> — storage for INT4_ARRAY value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextInt4Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>int4_array_return</i> was null, or <i>array_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextInt4Array retrieves the value of the current field of a message, which must be an INT4_ARRAY (array of four-byte integers) field. If TipcMsgNextInt4Array succeeds, it advances the current field to the next field in the message. TipcMsgNextInt4Array converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextInt4Array returns FALSE, it does not store values in <i>int4_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>int4_array_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendInt4Array, TipcMsgAppendInt4ArrayPtr

Examples This example reads an INT4_ARRAY field from a message and prints the values in the array:

```
T_INT4 *int4_array;
T_INT4 array_size;
T_INT4 counter;

if (!TipcMsgNextInt4Array(msg, &int4_array, &array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    TutOut("int4_array[%d] = %d\n", counter, int4_array[counter]);
}
```


TipcMsgNextInt8

Name	TipcMsgNextInt8 — get an INT8 field from a message
Synopsis	<pre>T_BOOL TipcMsgNextInt8(msg, int8_return) T_IPC_MSG msg; T_INT8 *int8_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>int8_return</i> — storage for INT8 value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextInt8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>int8_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextInt8 retrieves the value of the current field of a message, which must be an INT8 (eight-byte integer) field. If TipcMsgNextInt8 succeeds, it advances the current field to the next field in the message. TipcMsgNextInt8 converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextInt8 returns FALSE, it does not store a value in <i>int8_return</i>. Eight-byte integers are not available as native integer data types on all platforms.
See Also	TipcMsgAppendInt8

Examples This example creates a message, appends an INT8 field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT8 int8_val;

#define USER_MT_INT8_TEST 100
mt = TipcMtCreate("int8_test", USER_MT_INT8_TEST, "int8");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendInt8(msg, 82000000000)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextInt8(msg, &int8_val)) {
    return; /* error */
}
TutOut("int8_val == " T_INT8_SPEC "\n", int8_val);
```

TipcMsgNextInt8Array

Name	TipcMsgNextInt8Array — get an INT8_ARRAY field from a message
Synopsis	<pre>T_BOOL TipcMsgNextInt8Array(msg, int8_array_return, array_size_return) T_IPC_MSG msg; T_INT8 **int8_array_return; T_INT8 *array_size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>int8_array_return</i> — storage for INT8_ARRAY value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextInt8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>int8_array_return</i> was null, or <i>array_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextInt8Array retrieves the value of the current field of a message, which must be an INT8_ARRAY (array of eight-byte integers) field. If TipcMsgNextInt8Array succeeds, it advances the current field to the next field in the message. TipcMsgNextInt8Array converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextInt8Array returns FALSE, it does not store values in <i>int8_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>int8_array_return</i>. It points directly into an internal data structure. Eight-byte integers are not available as native integer data types on all platforms.
See Also	TipcMsgAppendInt8Array, TipcMsgAppendInt8ArrayPtr

Examples This example reads an INT8_ARRAY field from a message and prints the values in the array:

```
T_INT8 *int8_array;
T_INT8 array_size;
T_INT4 counter;

if (!TipcMsgNextInt8Array(msg, &int8_array, &array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    TutOut("int8_array[%d] = " T_INT8_SPEC "\n", counter,
          int8_array[counter]);
}
```

TipcMsgNextMsg

Name	TipcMsgNextMsg — get a MSG field from a message
Synopsis	<pre>T_BOOL TipcMsgNextMsg(msg, msg_return) T_IPC_MSG msg; T_IPC_MSG *msg_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>msg_return</i> — storage for MSG value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextMsg fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>msg_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextMsg retrieves the value of the current field of a message, which must be a MSG field. If TipcMsgNextMsg succeeds, it advances the current field to the next field in the message. TipcMsgNextMsg converts the field into the data formats used by this process if necessary.
Caution	If TipcMsgNextMsg returns FALSE, it does not store a value in <i>msg_return</i>. Do not modify the value stored in <i>msg_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendMsg, TipcMsgAppendMsgPtr

Examples This example creates a message, appends a MSG field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_IPC_MSG msg_val;

#define USER_MT_CONTAINER 100
mt = TipcMtCreate("container", USER_MT_CONTAINER, "msg");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}
msg_val = TipcMsgCreate(mt);
if (msg_val == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg_val, "speed_limit", 55.0)) {
    return; /* error */
}
if (!TipcMsgAppendMsg(msg, msg_val)) {
    return; /* error */
}
if (!TipcMsgDestroy(msg_val)) {
    return; /* error */
}
msg_val = NULL;

if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextMsg(msg, &msg_val)) {
    return; /* error */
}

if (!TipcMsgPrint(msg_val, TutOut)) {
    return; /* error */
}
```

TipcMsgNextMsgArray

Name	TipcMsgNextMsgArray — get a field containing an array of MSG fields from a message
Synopsis	<pre>T_BOOL TipcMsgNextMsgArray(msg, msg_array_return, array_size_return) T_IPC_MSG msg; T_IPC_MSG **msg_array_return; T_INT4 *array_size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>msg_array_return</i> — storage for MSG_ARRAY value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextMsgArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>msg_array_return</i> was null, or <i>array_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextMsgArray retrieves the value of the current field of a message, which must be a MSG_ARRAY field. If TipcMsgNextMsgArray succeeds, it advances the current field to the next field in the message. TipcMsgNextMsgArray converts the field into the data formats used by this process if necessary.
Caution	If TipcMsgNextMsgArray returns FALSE, it does not store values in <i>msg_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>msg_array_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendMsgArray, TipcMsgAppendMsgArrayPtr

Examples This example reads a MSG_ARRAY field from a message and prints the values in the array:

```
T_IPC_MSG *msg_array;
T_INT4 array_size;
T_INT4 counter;

if (!TipcMsgNextMsgArray(msg, &msg_array, &array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    if (!TipcMsgPrint(msg_array[counter], TutOut)) {
        return; /* error */
    }
}
```


TipcMsgNextReal4

Name	TipcMsgNextReal4 — get a REAL4 field from a message
Synopsis	<pre>T_BOOL TipcMsgNextReal4(msg, real4_return) T_IPC_MSG msg; T_REAL4 *real4_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>real4_return</i> — storage for REAL4 value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextReal4 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>real4_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextReal4 retrieves the value of the current field of a message, which must be a REAL4 (four-byte real number) field. If TipcMsgNextReal4 succeeds, it advances the current field to the next field in the message.
Caution	If TipcMsgNextReal4 returns FALSE, it does not store a value in <i>real4_return</i>. TipcMsgNextReal4 converts the field into the data format used by this process if necessary.
See Also	TipcMsgAppendReal4

Examples This example creates a message, appends a REAL4 field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL4 real4_val;

#define USER_MT_REAL4_TEST 100
mt = TipcMtCreate("real4_test", USER_MT_REAL4_TEST, "real4");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendReal4(msg, 42.0)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextReal4(msg, &real4_val)) {
    return; /* error */
}
TutOut("real4_val = %s\n", TutRealToStr(real4_val));
```

TpcMsgNextReal4Array

Name	TpcMsgNextReal4Array — get a field containing an array of REAL4 fields from a message
Synopsis	<pre> T_BOOL TpcMsgNextReal4Array(msg, real4_array_return, array_size_return) T_IPC_MSG msg; T_REAL4 **real4_array_return; T_INT4 *array_size_return; </pre>
Arguments	<i>msg</i> — message to get field from <i>real4_array_return</i> — storage for REAL4_ARRAY value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TpcMsgNextReal4Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>real4_array_return</i> was null, or <i>array_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TpcMsgNextReal4Array retrieves the value of the current field of a message, which must be a REAL4_ARRAY (array of four-byte real numbers) field. If TpcMsgNextReal4Array succeeds, it advances the current field to the next field in the message. TpcMsgNextReal4Array converts the field into the data format used by this process if necessary.
Caution	If TpcMsgNextReal4Array returns FALSE, it does not store values in <i>real4_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>real4_array_return</i>. It points directly into an internal data structure.
See Also	TpcMsgAppendReal4Array, TpcMsgAppendReal4ArrayPtr

Examples This example reads a REAL4_ARRAY field from a message and prints the values in the array:

```
T_REAL4 *real4_array;
T_INT4 array_size;
T_INT4 counter;

if (!TipcMsgNextReal4Array(msg, &real4_array, &array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    TutOut("real4_array[%d] = %s\n", counter,
          TutRealToStr(real4_array[counter]));
}
```

TipcMsgNextReal8

Name	TipcMsgNextReal8 — get a REAL8 field from a message
Synopsis	<pre>T_BOOL TipcMsgNextReal8(msg, real8_return) T_IPC_MSG msg; T_REAL8 *real8_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>real8_return</i> — storage for REAL8 value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextReal8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>real8_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextReal8 retrieves the value of the current field of a message, which must be a REAL8 (eight-byte real number) field. If TipcMsgNextReal8 succeeds, it advances the current field to the next field in the message. TipcMsgNextReal8 converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextReal8 returns FALSE, it does not store a value in <i>real8_return</i> .
See Also	TipcMsgAppendReal8

Examples This example creates a message, appends a REAL8 field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL8 real8_val;

#define USER_MT_REAL8_TEST 100
mt = TipcMtCreate("real8_test", USER_MT_REAL8_TEST, "real8");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendReal8(msg, 42.0)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextReal8(msg, &real8_val)) {
    return; /* error */
}
TutOut("real8_val = %s\n", TutRealToStr(real8_val));
```

TipcMsgNextReal8Array

Name	TipcMsgNextReal8Array — get a field containing an array of REAL8 fields from a message
Synopsis	<pre>T_BOOL TipcMsgNextReal8Array(msg, real8_array_return, array_size_return) T_IPC_MSG msg; T_REAL8 **real8_array_return; T_INT4 *array_size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>real8_array_return</i> — storage for REAL8_ARRAY value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextReal8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>real8_array_return</i> was null, or <i>array_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextReal8Array retrieves the value of the current field of a message, which must be a REAL8_ARRAY (array of eight-byte real numbers) field. If TipcMsgNextReal8Array succeeds, it advances the current field to the next field in the message. TipcMsgNextReal8Array converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextReal8Array returns FALSE, it does not store values in <i>real8_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>real8_array_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendReal8Array, TipcMsgAppendReal8ArrayPtr

Examples This example reads a REAL8_ARRAY field from a message and prints the values in the array:

```
T_REAL8 *real8_array;
T_INT4 array_size;
T_INT4 counter;

if (!TipcMsgNextReal8Array(msg, &real8_array, &array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    TutOut("real8_array[%d] = %s\n", counter,
          TutRealToStr(real8_array[counter]));
}
```


TipcMsgNextReal16

Name	TipcMsgNextReal16 — get a REAL16 field from a message
Synopsis	<pre>T_BOOL TipcMsgNextReal16(msg, real16_return) T_IPC_MSG msg; T_REAL16 *real16_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>real16_return</i> — storage for REAL16 value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextReal16 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>real16_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextReal16 retrieves the value of the current field of a message, which must be a REAL16 (16-byte real number) field. If TipcMsgNextReal16 succeeds, it advances the current field to the next field in the message. TipcMsgNextReal16 converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextReal16 returns FALSE, it does not store a value in <i>real16_return</i> .
	Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.
See Also	TipcMsgAppendReal16

Examples This example creates a message, appends a REAL16 field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL16 real16_val;

#define USER_MT_REAL16_TEST 100
mt = TipcMtCreate("real16_test", USER_MT_REAL16_TEST, "real16");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendReal16(msg, 1e1000)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextReal16(msg, &real16_val)) {
    return; /* error */
}
TutOut("real16_val = " T_REAL16_SPEC "\n", real16_val);
```

TipcMsgNextReal16Array

Name	TipcMsgNextReal16Array — get a field containing an array of REAL16 fields from a message
Synopsis	<pre>T_BOOL TipcMsgNextReal16Array(msg, real16_array_return, array_size_return) T_IPC_MSG msg; T_REAL16 **real16_array_return; T_INT4 *array_size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>real16_array_return</i> — storage for REAL16_ARRAY value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextReal16Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>real16_array_return</i> was null, or <i>array_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextReal16Array retrieves the value of the current field of a message, which must be a REAL16_ARRAY (array of 16-byte real numbers) field. If TipcMsgNextReal16Array succeeds, it advances the current field to the next field in the message. TipcMsgNextReal16Array converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextReal16Array returns FALSE, it does not store values in <i>real16_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>real16_array_return</i>. It points directly into an internal data structure.



Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.

See Also TipcMsgAppendReal16Array, TipcMsgAppendReal16ArrayPtr

Examples This example reads a REAL16_ARRAY field from a message and prints the values in the array:

```
T_REAL16 *real16_array;
T_INT4 array_size;
T_INT4 counter;

if (!TipcMsgNextReal16Array(msg, &real16_array, &array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    TutOut("real16_array[%d] = " T_REAL16_SPEC "\n", counter,
          real16_array[counter]);
}
```

TipcMsgNextStr

Name	TipcMsgNextStr — get a STR field from a message
Synopsis	<pre>T_BOOL TipcMsgNextStr(msg, str_return) T_IPC_MSG msg; T_STR *str_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>str_return</i> — storage for STR value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextStr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>str_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextStr retrieves the value of the current field of a message, which must be a STR (null-terminated character string) field. If TipcMsgNextStr succeeds, it advances the current field to the next field in the message. TipcMsgNextStr converts the field into the data format used by this process, if necessary.
Caution	If TipcMsgNextStr returns FALSE, it does not store a value in <i>str_return</i>. Do not modify the value stored in <i>str_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendStr, TipcMsgAppendStrPtr

Examples This example creates a message, appends a STR field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR str_val;

#define USER_MT_STR_TEST 100
mt = TipcMtCreate("str_test", USER_MT_STR_TEST, "str");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStr(msg, "Now is the time")) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextStr(msg, &str_val)) {
    return; /* error */
}
TutOut("str_val == %s\n", str_val);
```

TipcMsgNextStrArray

Name	TipcMsgNextStrArray — get a field containing an array of STR fields from a message
Synopsis	<pre>T_BOOL TipcMsgNextStrArray(msg, str_array_return, array_size_return) T_IPC_MSG msg; T_STR **str_array_return; T_INT4 *array_size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>str_array_return</i> — storage for STR_ARRAY value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextStrArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>str_array_return</i> was null, or <i>array_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextStrArray retrieves the value of the current field of a message, which must be a STR_ARRAY (array of null-terminated character strings) field. If TipcMsgNextStrArray succeeds, it advances the current field to the next field in the message. TipcMsgNextStrArray converts the field into the data format used by this process, if necessary.
Caution	If TipcMsgNextStrArray returns FALSE, it does not store values in <i>str_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>str_array_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendStrArray, TipcMsgAppendStrArrayPtr

Examples This example reads a STR_ARRAY field from a message and prints the values in the array:

```
T_STR *str_array;
T_INT4 array_size;
T_INT4 counter;

if (!TipcMsgNextStrArray(msg, &str_array, &array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    TutOut("str_array[%d] = %s\n", counter, str_array[counter]);
}
```


TipcMsgNextStrReal8

Name	TipcMsgNextStrReal8 — get a STR field and a REAL8 field from a message
Synopsis	<pre>T_BOOL TipcMsgNextStrReal8(msg, str_return, real8_return) T_IPC_MSG msg; T_STR *str_return; T_REAL8 *real8_return;</pre>
Arguments	<i>msg</i> — message to get fields from <i>str_return</i> — storage for STR value from field <i>real8_return</i> — storage for REAL8 value from field
Return Values	TRUE if the fields were successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextStrReal8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>str_return</i> was null, or <i>real8_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextStrReal8 is a convenience function that retrieves both a STR field and a REAL8 field from a message. TipcMsgNextStrReal8 is useful for accessing NUMERIC_DATA messages. The NUMERIC_DATA message type is often the most frequently used message type in SmartSockets. TipcMsgNextStrReal8 converts the field into the data formats used by this process if necessary.
Caution	If TipcMsgNextStrReal8 returns FALSE, it does not store values in <i>str_return</i> and <i>real8_return</i>. Do not modify the value stored in <i>str_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendStrReal8

Examples This example creates a message, appends STR and REAL8 fields, and reads the fields back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR str_val;
T_REAL8 real8_val;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "speed_limit", 55.0)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextStrReal8(msg, &str_val, &real8_val)) {
    return; /* error */
}
TutOut("str_val = %s, real8_val = %s\n",
       str_val, TutRealToStr(real8_val));
```

TipcMsgNextTimestamp

Name	TipcMsgNextTimestamp — get a TIMESTAMP field from a message
Synopsis	<pre>T_BOOL TipcMsgNextTimestamp(msg, timestamp_return) T_IPC_MSG msg; T_REAL8 *timestamp_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>timestamp_return</i> — storage for TIMESTAMP value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextTimestamp fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>timestamp_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextTimestamp retrieves the value of the current field of a message, which must be a TIMESTAMP (eight-byte real number) field. A timestamp is the current computer wall clock time in the number of seconds since midnight January 1, 1970. If TipcMsgNextTimestamp succeeds, it advances the current field to the next field in the message. TipcMsgNextTimestamp converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextTimestamp returns FALSE, it does not store a value in <i>timestamp_return</i> .
See Also	TipcMsgAppendTimestamp

Examples This example creates a message, appends a `TIMESTAMP` field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_REAL8 timestamp_val;

#define USER_MT_TIMESTAMP_TEST 100
mt = TipcMtCreate("timestamp_test", USER_MT_TIMESTAMP_TEST,
                 "timestamp");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendTimestamp(msg, TutGetWallTime())) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextTimestamp(msg, &timestamp_val)) {
    return; /* error */
}
TutOut("timestamp_val = %s\n", TutRealToStr(timestamp_val));
```

TipcMsgNextTimestampArray

Name	TipcMsgNextTimestampArray — get a field containing an array of <code>TIMESTAMP</code> fields from a message
Synopsis	<pre>T_BOOL TipcMsgNextTimestampArray(msg, timestamp_array_return, array_size_return) T_IPC_MSG msg; T_REAL8 **timestamp_array_return; T_INT4 *array_size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>timestamp_array_return</i> — storage for <code>TIMESTAMP_ARRAY</code> value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextTimestampArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_NULL_PTR</code> — <i>msg</i> was null, <i>timestamp_array_return</i> was null, or <i>array_size_return</i> was null • <code>T_ERR_MSG_INVALID</code> — <i>msg</i> was not a valid message • <code>T_ERR_MSG_EOM</code> — the current field is at the end of message • <code>T_ERR_FT_MISMATCH</code> — the current field is of the wrong type • <code>T_ERR_VAL_UNKNOWN</code> — the current field is of unknown value
Description	TipcMsgNextTimestampArray retrieves the value of the current field of a message, which must be a <code>TIMESTAMP_ARRAY</code> (array of eight-byte real numbers) field. A timestamp is the current computer wall clock time in the number of seconds since midnight January 1, 1970. If TipcMsgNextTimestampArray succeeds, it advances the current field to the next field in the message. TipcMsgNextTimestampArray converts the field into the data format used by this process if necessary.
Caution	If TipcMsgNextTimestampArray returns FALSE, it does not store values in <i>timestamp_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>timestamp_array_return</i>. It points directly into an internal data structure.

See Also TipcMsgAppendTimestampArray, TipcMsgAppendTimestampArrayPtr

Examples This example reads a `TIMESTAMP_ARRAY` field from a message and prints the values in the array:

```
T_REAL8 *timestamp_array;
T_INT4 array_size;
T_INT4 counter;

if (!TipcMsgNextTimestampArray(msg, &timestamp_array,
&array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    TutOut("timestamp_array[%d] = %s\n", counter,
          TutRealToStr(timestamp_array[counter]));
}
```

TipcMsgNextType

Name	TipcMsgNextType — get the field type of the current field of a message
Synopsis	<pre>T_BOOL TipcMsgNextType(msg, ft_return) T_IPC_MSG msg; T_IPC_FT *ft_return;</pre>
Arguments	<i>msg</i> — message to get field type from <i>ft_return</i> — storage for field type
Return Values	TRUE if the field type was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextType fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>ft_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message
Description	TipcMsgNextType retrieves the type of the current field of a message. TipcMsgNextType does not advance the current field to the next field in the message.
Caution	If TipcMsgNextType returns FALSE, it does not store a value in <i>ft_return</i> .
See Also	TipcMsgNext* (TipcMsgNextBinary through TipcMsgNextXml), TipcMsgGetType

Examples This example creates a message, appends a REAL8 field, and reads the field type back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_IPC_FT ft;

#define USER_MT_REAL8_TEST 100
mt = TipcMtCreate("real8_test", USER_MT_REAL8_TEST, "real8");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendReal8(msg, 42.0)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextType(msg, &ft)) {
    return; /* error */
}
TutOut("ft == %d\n", ft);
```


TipcMsgNextUnknown

Name	TipcMsgNextUnknown — checks if the next field in a message is of unknown value
Synopsis	<pre>T_BOOL TipcMsgNextUnknown(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to check field from
Return Values	TRUE if the field was successfully checked and was of value unknown, FALSE otherwise.
Diagnostics	If TipcMsgNextUnknown fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_VAL_MISMATCH — the current field is not of unknown value
Description	TipcMsgNextUnknown checks to see if the current field of a message is of known value. If the <i>unknown_value</i> property of a field is set, TipcMsgNextUnknown succeeds and advances the current field to the next field in the message. If the <i>unknown_value</i> property is not set, TipcMsgNextUnknown fails and does not advance to the next field in the message, allowing the application to use a different function to retrieve that value from the message. Frequently, messages have a fixed number of fields in a constant order. In that case, omitting a field when data occasionally does not exist is not feasible. Setting the <i>unknown_value</i> property allows fields to be marked as having no data. This feature is most useful when the entire range of values is valid within an application and there is a need to denote that the field has no valid data.
Caution	A FALSE return value can indicate several things. Check the value of the global SmartSockets error number before assuming that it means the current field is valid.
See Also	TipcMsgAppendUnknown, TipcMsgGetCurrentFieldKnown

Examples This example creates a message, appends a field of value unknown, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
mt = TipcMtLookupByNum (T_MT_NUMERIC_DATA);
if (mt==NULL) {
    return; /* error */
}

msg=TipcMsgCreate(mt);
if (msg==NULL) {
    return; /* error */
}

if (!TipcMsgAppendUnknown(msg, T_IPC_FT_INT4)) {
    return; /* error */
}

if (!TipcMsgSetCurrent(msg,0)) {
    return; /* error */
}

if (!TipcMsgNextUnknown(msg)) {
    return; /* error */
}
```

TipcMsgNextUtf8

Name	TipcMsgNextUtf8 — get a UTF8 field from a message
Synopsis	<pre>T_BOOL TipcMsgNextUtf8(msg, utf8_return) T_IPC_MSG msg; T_STR *utf8_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>utf8_return</i> — storage for UTF8 value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextUtf8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>utf8_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextUtf8 retrieves the value of the current field of a message, which must be a UTF8 field. If TipcMsgNextUtf8 succeeds, it advances the current field to the next field in the message. TipcMsgNextUtf8 converts the field into the data format used by this process, if necessary.
Caution	If TipcMsgNextUtf8 returns FALSE, it does not store a value in <i>utf8_return</i>. Do not modify the value stored in <i>utf8_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendUtf8, TipcMsgAppendUtf8Ptr

Examples This example creates a message, appends an INT8 field, and reads the field back from the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR utf8_val;

#define USER_MT_UTF8_TEST 100
mt = TipcMtCreate("utf8_test", USER_MT_UTF8_TEST, "utf8");
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendUtf8(msg, 82000000000)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextUtf8(msg, &utf8_val)) {
    return; /* error */
}
TutOut("utf8_val == " T_UTF8_SPEC "\n", utf8_val);
```

TpcMsgNextUtf8Array

Name	TpcMsgNextUtf8Array — get a field containing an array of UTF8 fields from a message
Synopsis	<pre>T_BOOL TpcMsgNextUtf8Array(msg, utf8_array_return, array_size_return) T_IPC_MSG msg; T_STR **utf8_array_return; T_INT4 *array_size_return;</pre>
Arguments	<i>msg</i> — message to get field from <i>utf8_array_return</i> — storage for UTF8_ARRAY value from field <i>array_size_return</i> — storage for number of elements in array
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TpcMsgNextUtf8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null, <i>utf8_array_return</i> was null, or <i>array_size_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TpcMsgNextUtf8Array retrieves the value of the current field of a message, which must be a UTF8_ARRAY (array of UTF8 values) field. If TpcMsgNextUtf8Array succeeds, it advances the current field to the next field in the message. TpcMsgNextUtf8Array converts the field into the data format used by this process, if necessary.
Caution	If TpcMsgNextUtf8Array returns FALSE, it does not store values in <i>utf8_array_return</i> and <i>array_size_return</i>. Do not modify the value stored in <i>utf8_array_return</i>. It points directly into an internal data structure.
See Also	TpcMsgAppendUtf8Array, TpcMsgAppendUtf8ArrayPtr

Examples This example reads a UTF8_ARRAY field from a message and prints the values in the array:

```
T_STR *utf8_array;
T_STR array_size;
T_INT4 counter;

if (!TipcMsgNextutf8Array(msg, &utf8_array, &array_size)) {
    return; /* error */
}
TutOut("array_size = %d\n", array_size);

for (counter = 0; counter < array_size; counter++) {
    TutOut("utf8_array[%d] = " T_UTF8_SPEC "\n", counter,
          utf8_array[counter]);
}
```

TipcMsgNextXml

Name	TipcMsgNextXml — get the next XML string from a message
Synopsis	<pre>T_BOOL TipcMsgNextXml(msg, xml_return) T_IPC_MSG msg; T_XML xml_return;</pre>
Arguments	<i>msg</i> — message from which to get the field <i>xml_return</i> — storage for XML value from field
Return Values	TRUE if the field was successfully retrieved from the message, FALSE otherwise.
Diagnostics	If TipcMsgNextXml fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>xml_return</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_FT_MISMATCH — the current field is of the wrong type • T_ERR_VAL_UNKNOWN — the current field is of unknown value
Description	TipcMsgNextXml retrieves the value of the current field of a message, which must be an XML field. If TipcMsgNextXml succeeds, it advances the current field to the next field in the message.
Caution	If TipcmsgNextXml returns FALSE, it does not store a value in <i>xml_return</i>. Do not modify the value stored in <i>xml_return</i>. It points directly into an internal data structure.
See Also	TipcMsgAppendXml; see the <i>TIBCO SmartSockets Utilities</i> for information on TutXmlCreate.

Examples

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_XML xml, xml_val;

T_XML xml_val;
#define USER_MT_XML_TEST 100
mt = TipcMtCreate("xml_test", USER_MT_XML_TEST, "xml");
if (mt == NULL) {
    return; /* error */
}
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}
xml = TutXmlCreate("<data><data1><temp=\"hot\"/></data1></data>");
if (xml == NULL) {
    return; /* error */
}
if (!TipcMsgAppendXml(msg, xml)) {
    return; /* error */
}
if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}
if (!TipcMsgNextXml(msg, &xml_val)) {
    return; /* error */
}

```


TipcMsgPrint

Name	TipcMsgPrint — print all information about of a message in portable, reproducible format
Synopsis	<pre>T_BOOL TipcMsgPrint(msg, func) T_IPC_MSG msg; T_OUT_FUNC func;</pre>
Arguments	<i>msg</i> — message to print <i>func</i> — TutOut-style output function
Return Values	TRUE if the message was successfully printed, FALSE otherwise.
Diagnostics	If TipcMsgPrint fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>func</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgPrint prints all information about a message in a portable, reproducible format to a TutOut-style output function. The output function is called once for each line of output. TipcMsgPrint first prints the non-data properties of the message, such as the type and destination. TipcMsgPrint then prints the message data, two lines for each field. The first lines shows the field type, and the second line the field value.
Caution	None
See Also	TutOut (see the <i>TIBCO SmartSockets Utilities</i>)

Examples This example creates a message, appends fields, and prints the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;

mt = TpcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TpcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TpcMsgAppendStrReal8(msg, "speed_limit", 65.0)) {
    return; /* error */
}

if (!TpcMsgPrint(msg, TutOut)) {
    return; /* error */
}
```

The output from the above example is similar to:

```
type = numeric_data
sender = <(NULL)>
dest = <(NULL)>
max = 2048
size = 44
current = 0
read_only = false
priority = 0
delivery_mode = best_effort
ref_count = 1
seq_num = 0
resend_mode = false
user_prop = 0
data (num_fields = 2):
    str "speed_limit"
    real8 65
```

TipcMsgPrintError

Name	TipcMsgPrintError — report an error about an unexpected message
Synopsis	<pre>T_BOOL TipcMsgPrintError(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message for error report
Return Values	TRUE if the error report was successfully output, FALSE otherwise.
Diagnostics	If TipcMsgPrintError fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcMsgPrintError is used to report an error for an unexpected message. An unexpected message is one that has no connection process callbacks for the message's type. Call TipcMsgPrintError with a connection default callback, as these callbacks are called only for message types without any process callbacks. The first line of the error report has the form: <pre>WARNING: Received unwanted message at time <i>time</i>.</pre> The time printed is the current data time from TutGetCurrentTime, and it is printed using the format specified in the option Time_Format. TipcMsgPrintError then prints the contents of the message with TipcMsgPrint.
Caution	None
See Also	TipcConnDefaultCbCreate, TipcSrvDefaultCbCreate

Examples This example creates a message default callback that uses TipcMsgPrintError to inform the user of an unexpected message from RTserver:

```
static void T_ENTRY default_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_DEFAULT_CB_DATA data;
T_CB_ARG arg;
{
    if (!TipcMsgPrintError(data->msg)) {
        return; /* error */
    }
} /* default_cb */

/* ===== */
/*...code from calling function is below */
if (TipcSrvDefaultCbCreate(default_cb, NULL)) {
    return; /* error */
}
```

TipcMsgRead

Name	TipcMsgRead — read fields from a message
Synopsis	<pre>T_BOOL TipcMsgRead(msg, ...) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to read fields from additional arguments — null-terminated field type-value pairs
Return Values	TRUE if all the fields were successfully read, FALSE otherwise.
Diagnostics	If TipcMsgRead fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_INVALID — a field type argument was not a valid field type • T_ERR_MSG_EOM — the current field is at the end of message • any error number from TipcMsgReadVa
Description	TipcMsgRead is a convenience function that takes a null-terminated variable number of field type-value pairs and reads these fields from the message. The field type part of the pair is a T_IPC_FT, and the value part of the pair corresponds to the argument(s) for the related TipcMsgNext* function (for example, if the type part is T_IPC_FT_REAL8, then the value part is a pointer to a REAL8 number) or TipcMsgGet* function (for example, if the type part is T_IPC_PROP_DELIVERY_MODE, then the value part is a pointer to a T_IPC_DELIVERY_MODE setting). TipcMsgRead is a short wrapper function that calls TipcMsgReadVa.
Caution	The last argument passed to TipcMsgRead must be null, otherwise TipcMsgRead may read garbage fields from the message. Using TipcMsgRead is slightly slower than using the TipcMsgNext* functions directly.
See Also	TipcMsgUpdateNamedBinary, TipcMsgReadVa

Examples This example creates a message, appends two fields, and reads the fields back with TipcMsgRead:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR var_name;
T_REAL8 var_value;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "speed_limit", 55.0)) {
    return; /* error */
}

if (!TipcMsgRead(msg,
                 T_IPC_FT_STR, &var_name,
                 T_IPC_FT_REAL8, &var_value,
                 NULL)) {
    return; /* error */
}
```

TipcMsgReadVa

Name	TipcMsgReadVa — read fields from a message (<i>va_list</i> version)
Synopsis	<pre>T_BOOL TipcMsgReadVa(msg, var_arg_list) T_IPC_MSG msg; va_list var_arg_list;</pre>
Arguments	<i>msg</i> — message to read fields from <i>var_arg_list</i> — argument list
Return Values	TRUE if all the fields were successfully read, FALSE otherwise.
Diagnostics	If TipcMsgReadVa fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_INVALID — a field type argument was not a valid field type • T_ERR_MSG_EOM — the current field is at the end of message • T_ERR_VAL_UNKNOWN — the current field is of value unknown • any error number from TipcMsgNext*
Description	TipcMsgReadVa is the non-varargs version of TipcMsgRead. (TipcMsgRead actually calls TipcMsgReadVa.) TipcMsgReadVa can be used to implement a varargs function. In the C language, a varargs function (one that takes a variable number of arguments) cannot call another varargs function with the original arguments. A varargs function can (and usually does) call a helper function that takes a fixed number of arguments, with the last argument usually of type <i>va_list</i>. TipcMsgReadVa is TipcMsgRead's helper function.
Caution	None
See Also	TipcMsgRead, TipcMsgUpdateNamedBinary, TipcMsgNext* (TipcMsgNextBinary through TipcMsgNextXml)

Examples This example shows how to write a varargs function in ANSI C that reads fields from a message and then destroys the message:

```
T_BOOL my_msg_read_destroy(T_IPC_MSG msg,
                          ...)
{
    va_list var_arg_list;
    int read_status;
    int destroy_status;

    va_start(var_arg_list, msg);
    read_status = TipcMsgReadVa(msg, var_arg_list);
    va_end(var_arg_list);
    destroy_status = TipcMsgDestroy(msg);

    /* return FALSE if either status is FALSE */
    return read_status && destroy_status;
} /* my_msg_read_destroy */
```


TipcMsgSetArrivalTimestamp

Name	TipcMsgSetArrivalTimestamp — set the arrival timestamp property of a message
Synopsis	<pre>T_BOOL TipcMsgSetArrivalTimestamp(msg, arrival_timestamp) T_IPC_MSG msg; T_REAL8 arrival_timestamp;</pre>
Arguments	<i>msg</i> — message to set arrival timestamp for <i>arrival_timestamp</i> — message arrival timestamp
Return Values	TRUE if the arrival timestamp was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetArrivalTimestamp fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>arrival_timestamp</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_TOO_SMALL — <i>arrival_timestamp</i> was negative
Description	TipcMsgSetArrivalTimestamp sets the arrival timestamp property of a message. As messages are read, their arrival timestamp property will automatically be set. A value of 0.0 indicates that the arrival timestamp property was not set.
Caution	None
See Also	TipcMsgGetArrivalTimestamp
Examples	This example sets the arrival timestamp property of a message to the current time: <pre>if (!TipcMsgSetArrivalTimestamp(msg, TutGetWallTime())) { return; /* error */ }</pre>

TipcMsgSetCompression

Name	TipcMsgSetCompression — enable or disable message compression
Synopsis	<pre>T_BOOL TipcMsgSetCompression(msg, compress) T_IPC_MSG msg; T_BOOL compress;</pre>
Arguments	<i>msg</i> — message to set compression for <i>compress</i> — message compression status (TRUE to enable compression, FALSE to disable)
Return Values	TRUE if the compression status was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetCompression fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_INVALID – <i>compress</i> was not TRUE or FALSE
Description	TipcMsgSetCompression sets the compression status of a message. When the compression status is set to TRUE, the fields in the message are automatically compressed when the message is sent. The compression is performed using the algorithm specified by the Compression_Name option. The message is automatically decompressed when a receiver of the message attempts to access any of the message’s fields. TipcMsgSetCompression overwrites the default compression status set by TipcMtSetCompression.
Caution	Do not send compressed messages to an RT process running on a SmartSockets software release lower than 6.5.0. Sending a compressed message to an RT process running an earlier version of SmartSockets may result in unexpected behavior. Typically, the RT process running on an earlier version of SmartSockets returns an error code of T_ERR_FT_MISMATCH or T_ERR_FT_INVALID when attempting to access the fields of a compressed message. However, it is also possible for the RT process to retrieve invalid data from the field without generating an error.

See Also `TipcMsgGetCompression`, `TipcMtSetCompression`

Examples This example enables compression of a message:

```
if (!TipcMsgSetCompression(msg, T_TRUE)) {  
    return; /* error */  
}
```

TipcMsgSetCorrelationId

Name	TipcMsgSetCorrelationId — set the correlation identifier property of a message
Synopsis	<pre>T_BOOL TipcMsgSetCorrelationId(msg, correlation_id) T_IPC_MSG msg; T_STR correlation_id;</pre>
Arguments	<i>msg</i> — message to generate identifier for <i>correlation_id</i> — message correlation identifier
Return Values	TRUE if the identifier was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetCorrelationId fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>correlation_id</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgSetCorrelationId sets the correlation identifier property of a message. The correlation identifier property is a string whose meaning depends on how the message is being used. Applications of messages are free to use the correlation identifier for any purpose.
Caution	None
See Also	TipcMsgGetCorrelationId, TipcMsgGenerateMessageId, TipcMsgGetMessageId
Examples	This example sets the correlation identifier property of a message to the message identifier property of another message: <pre>T_STR id; if (!TipcMsgGetMessageId(request_msg, &id)) { return; /*error*/ } if (!TipcMsgSetCorrelationId(reply_msg, id)) { return; /*error*/ }</pre>

TpcMsgSetCurrent

Name	TpcMsgSetCurrent — set the current field of a message
Synopsis	<pre>T_BOOL TpcMsgSetCurrent(msg, field_num) T_IPC_MSG msg; T_INT4 field_num;</pre>
Arguments	<i>msg</i> — message to set current field for <i>field_num</i> — field number (0 indicates the first field)
Return Values	TRUE if the current field was successfully set for the message, FALSE otherwise.
Diagnostics	If TpcMsgSetCurrent fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — <i>field_num</i> was less than 0 or greater than the number of fields in the message minus 1
Description	TpcMsgSetCurrent sets the current field of a message. TpcMsgSetCurrent does not change the number of fields in the message. Any new fields are still appended to the message.
Caution	None
See Also	TpcMsgGetDeliveryMode, TpcMsgGetNumFields, TpcMsgSetNumFields, TpcMsgTraverse

Examples This example creates a message, appends fields, reads the fields, sets the current field to the first field, and reads the fields again:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR str;
T_REAL8 real8;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "speed_limit", 65.0)) {
    return; /* error */
}

if (!TipcMsgNextStrReal8(msg, &str, &real8)) {
    return; /* error */
}
TutOut("str = %s, real8 = %s.\n", str, TutRealToStr(real8));

if (!TipcMsgSetCurrent(msg, 0)) {
    return; /* error */
}

if (!TipcMsgNextStrReal8(msg, &str, &real8)) {
    return; /* error */
}
TutOut("str = %s, real8 = %s.\n", str, TutRealToStr(real8));
```

TipcMsgSetDeliveryMode

Name	TipcMsgSetDeliveryMode — set the delivery mode of a message
Synopsis	<pre>T_BOOL TipcMsgSetDeliveryMode(msg, delivery_mode) T_IPC_MSG msg; T_IPC_DELIVERY_MODE delivery_mode;</pre>
Arguments	<i>msg</i> — message to set delivery mode for <i>delivery_mode</i> — message delivery mode (T_IPC_DELIVERY_BEST_EFFORT, T_IPC_DELIVERY_ORDERED, T_IPC_DELIVERY_SOME, or T_IPC_DELIVERY_ALL)
Return Values	TRUE if the delivery mode was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetDeliveryMode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_INVALID — <i>delivery_mode</i> was not a valid T_IPC_DELIVERY_MODE
Description	TipcMsgSetDeliveryMode sets the delivery mode of a message. The delivery mode of a message controls what level of guarantee is used when the message is sent through a connection with TipcConnMsgSend. Valid delivery mode values are: • T_IPC_DELIVERY_BEST_EFFORT • T_IPC_DELIVERY_ORDERED • T_IPC_DELIVERY_SOME • T_IPC_DELIVERY_ALL TipcMsgSetDeliveryMode overwrites the default value of the delivery mode property set by TipcMtSetDeliveryMode. See TipcMsgGetDeliveryMode for more information on message delivery modes.
Caution	If you are sending messages to JMS clients, set the delivery mode to T_IPC_DELIVERY_ORDERED.

See Also TipcMsgGetDeliveryMode, TipcMtSetDeliveryMode

Examples This example sets the delivery mode of a message:

```
if (!TipcMsgSetDeliveryMode(msg, T_IPC_DELIVERY_ALL)) {  
    return; /* error */  
}
```


TipcMsgSetDeliveryTimeout

Name	TipcMsgSetDeliveryTimeout — set the delivery timeout of a message
Synopsis	<pre>T_BOOL TipcMsgSetDeliveryTimeout(<i>msg</i>, <i>delivery_timeout</i>) T_IPC_MSG <i>msg</i>; T_REAL8 <i>delivery_timeout</i>;</pre>
Arguments	<i>msg</i> — message to set the delivery timeout for <i>delivery_timeout</i> — message delivery timeout
Return Values	TRUE if the delivery timeout was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetDeliveryTimeout fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_TOO_SMALL — <i>delivery_timeout</i> was negative
Description	This function sets the delivery timeout of a message. The delivery timeout of a message controls how long to wait for guaranteed message delivery to complete when a message is sent through a connection with TipcConnMsgSend. TipcMsgSetDeliveryTimeout overwrites the default value of the delivery timeout property set by TipcMtSetDeliveryTimeout. See TipcMsgGetDeliveryTimeout for more information on message delivery timeouts.
Caution	None
See Also	TipcMsgGetDeliveryTimeout, TipcMtSetDeliveryTimeout
Examples	This example sets the delivery timeout of a message to 5 seconds: <pre>if (!TipcMsgSetDeliveryTimeout(msg, 5.0)) { return; /* error */ }</pre>

TpcMsgSetDest

Name TpcMsgSetDest — set the destination of a message

Synopsis `T_BOOL TpcMsgSetDest(msg, dest)`
`T_IPC_MSG msg;`
`T_STR dest;`

Arguments *msg* — message to set destination for
dest — message destination

Return Values TRUE if the destination was successfully set for the message, FALSE otherwise.

Diagnostics If TpcMsgSetDest fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *msg* was null or *dest* was null
- T_ERR_MSG_INVALID — *msg* was not a valid message
- T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
- T_ERR_VAL_INVALID — *dest* was not a valid subject

Description TpcMsgSetDest sets the destination property of a message. The destination property is a string whose meaning depends on how the message is being used. Other applications of messages are free to use the sender and destination for another purpose. Message sender and destination properties can be null if they are not needed. TpcMsgSetDest makes a copy of the message destination.

Messages sent with publish-subscribe between RTclients by RTserver, for example, use a subject name for the message sender and destination.

Caution TpcMsgSetDest does not detect errors in *dest* (such as misspelled subject names).

See Also TpcMsgGetDest, TpcMsgSetType

Examples This example sets the destination property of a message to `_all`:

```
if (!TpcMsgSetDest(msg, "_all")) {
    return; /* error */
}
```

TipcMsgSetExpiration

Name	TipcMsgSetExpiration — set the expiration property of a message
Synopsis	<pre>T_BOOL TipcMsgSetExpiration(msg, expiration) T_IPC_MSG msg; T_REAL8 expiration;</pre>
Arguments	<i>msg</i> — message to set the expiration of <i>expiration</i> — message expiration time
Return Values	TRUE if the expiration was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetExpiration fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>expiration</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_TOO_SMALL — <i>expiration</i> was negative
Description	TipcMsgSetExpiration sets the expiration, also known as time to live, property of a message. The expiration property controls how long, in seconds, the message should be in existence. Applications of messages are free to use the expiration for any purpose. A value of 0.0 indicates that the message does not expire.
Caution	None
See Also	TipcMsgGetExpiration
Examples	This example sets the expiration property of a message to 30 seconds: <pre>if (!TipcMsgSetExpiration(msg, 30.0)) { return; /* error */ }</pre>

TipcMsgSetHeaderStrEncode

Name	TipcMsgSetHeaderStrEncode — set the header string encode property of a message
Synopsis	<pre>T_BOOL TipcMsgSetHeaderStrEncode(msg, header_str_encode) T_IPC_MSG msg; T_BOOL header_str_encode;</pre>
Arguments	<i>msg</i> — message to set header string encode property for <i>header_str_encode</i> — message header string encode property
Return Values	TRUE if the header string encode property was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetHeaderStrEncode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_INVALID — <i>header_str_encode</i> was not TRUE or FALSE
Description	TipcMsgSetHeaderStrEncode sets the header string encode property of a message. The header string encode property of a message controls whether the message header string properties are converted into four-byte integers when the message is sent through a connection with TipcConnMsgSend. TipcMsgSetHeaderStrEncode overwrites the default value of the header string encode property set by TipcMtSetHeaderStrEncode. See TipcMsgGetHeaderStrEncode for more information on the message header string encode property.
Caution	None
See Also	TipcMsgGetHeaderStrEncode, TipcMtSetHeaderStrEncode
Examples	This example enables the header string encoding property of a message: <pre>if (!TipcMsgSetHeaderStrEncode(msg, TRUE)) { return; /* error */ }</pre>

TipcMsgSetLbMode

Name	TipcMsgSetLbMode — set the load balancing mode of a message
Synopsis	<pre>T_BOOL TipcMsgSetLbMode(msg, lb_mode) T_IPC_MSG msg; T_IPC_LB_MODE lb_mode;</pre>
Arguments	<i>msg</i> — message to set load balancing mode for <i>lb_mode</i> — message load balancing mode (T_IPC_LB_NONE, T_IPC_LB_ROUND_ROBIN, T_IPC_LB_WEIGHTED, or T_IPC_LB_SORTED)
Return Values	TRUE if the load balancing mode was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetLbMode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_INVALID — <i>lb_mode</i> was not a valid T_IPC_LB_MODE
Description	TipcMsgSetLbMode sets the load balancing mode of a message. The load balancing mode of a message controls how (or if) the message is load balanced between RTclients subscribed to a load balanced subject. Valid modes are T_IPC_LB_NONE, T_IPC_LB_ROUND_ROBIN, T_IPC_LB_WEIGHTED, and T_IPC_LB_SORTED. TipcMsgSetLbMode overwrites the default value of the load balancing mode property set by TipcMtSetLbMode. The default load balancing mode is T_IPC_LB_NONE. In this mode, no load balancing is done. The message is delivered to all RTclients subscribing to the subject to which the message is published. The other load balancing modes are T_IPC_LB_ROUND_ROBIN, T_IPC_LB_WEIGHTED, and T_IPC_LB_SORTED. In these modes, one and only one RTclient in the list of potential receivers receives the message. In ROUND_ROBIN mode, each successive message is sent to the next RTclient in a circular list of RTclients, to distribute the load evenly. WEIGHTED mode sends the message to the most responsive RTclient (as measured by the fewest pending GMD acknowledgments), reasoning that this is also the RTclient with the least load. In SORTED mode, each message is sent to the RTclient whose unique subject name sorts lowest lexicographically.

For more information on load balancing, see the *TIBCO SmartSockets User's Guide*.

Caution None

See Also TipcMsgGetLbMode, TipcMtGetLbMode, TipcMtSetLbMode

Examples This example sets the load balancing mode of a message to T_IPC_LB_WEIGHTED:

```
if (!TipcMsgSetLbMode(msg, T_IPC_LB_WEIGHTED)) {  
    return; /* error */  
}
```

TipcMsgSetNameCurrent

Name	TipcMsgSetNameCurrent — sets the name of the current field
Synopsis	<pre>T_BOOL TipcMsgSetNameCurrent(msg, name) T_IPC_MSG msg; T_STR name;</pre>
Arguments	<i>msg</i> — message containing field whose name is to be set <i>name</i> — name of the field
Return Values	TRUE if the set was successful, FALSE otherwise.
Diagnostics	If TipcMsgSetNameCurrent fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_ALREADY_EXISTS — the field name already exists in the message
Description	TipcMsgSetNameCurrent sets the name of the current field (the field that would be returned by a TipcMsgNext* function). If the current field has a name, it is overridden. If another field in the message already uses this name, this function fails.
Examples	This example sets the name of the current field: <pre>T_BOOL status; status = TipcMsgSetNameCurrent(msg, "foo"); if (status == FALSE) { return; /* error */ }</pre>

TipcMsgSetNumFields

Name	TipcMsgSetNumFields — set the number of fields in a message
Synopsis	<pre>T_BOOL TipcMsgSetNumFields(msg, num_fields) T_IPC_MSG msg; T_INT4 num_fields;</pre>
Arguments	<i>msg</i> — message to set number of fields for <i>num_fields</i> — number of fields
Return Values	TRUE if the number of fields was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetNumFields fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_TOO_SMALL — <i>num_fields</i> was less than 0 • T_ERR_VAL_TOO_LARGE — <i>num_fields</i> was greater than the number of fields in the message • any error number from TipcMsgTraverse
Description	TipcMsgSetNumFields sets the number of fields in a message by truncating the message to the desired number of fields. If <i>num_fields</i> is 0, the message is completely emptied (the message data buffer is not changed, but any fields in the data buffer are lost). TipcMsgSetNumFields is useful for reusing messages, rather than continually destroying and creating messages.
Caution	There is no way to recover the old fields in a message after calling TipcMsgSetNumFields.
See Also	TipcMsgSetCurrent, TipcMsgTraverse

Examples This example creates a message, appends fields, empties the message, and appends different fields:

```
T_IPC_MT mt;
T_IPC_MSG msg;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}
if (!TipcMsgAppendStrReal8(msg, "speed_limit", 65.0)) {
    return; /* error */
}

if (!TipcMsgSetNumFields(msg, 0)) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "speed_limit", 35.0)) {
    return; /* error */
}
```

TpcMsgSetPriority

Name	TpcMsgSetPriority — set the priority of a message
Synopsis	<pre>T_BOOL TpcMsgSetPriority(msg, priority) T_IPC_MSG msg; T_INT2 priority;</pre>
Arguments	<i>msg</i> — message to set priority for <i>priority</i> — message priority
Return Values	TRUE if the priority was successfully set for the message, FALSE otherwise.
Diagnostics	If TpcMsgSetPriority fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TpcMsgSetPriority sets the priority of a message. Message priorities are used by TpcConnRead to queue incoming messages in priority order. TpcMsgSetPriority overwrites the default value of the priority property set by TpcMtSetPriority. See TpcMsgGetPriority for more information on message priorities.
Caution	Setting message priorities on GMD or load-balanced (LB) messages is not recommended. GMD and LB rely on high sequence numbers for correct functioning, and unexpected behavior may occur if such messages are received out of order due to the use of message priorities.
See Also	TpcMsgGetPriority, TpcMtSetPriority
Examples	This example sets the priority of a message to 2: <pre>if (!TpcMsgSetPriority(msg, 2)) { return; /* error */ }</pre>

TipcMsgSetReplyTo

Name	TipcMsgSetReplyTo — set the reply to destination property of a message
Synopsis	<pre>T_BOOL TipcMsgSetReplyTo(msg, reply_to_dest) T_IPC_MSG msg; T_STR reply_to_dest;</pre>
Arguments	<i>msg</i> — message to set reply to destination for <i>reply_to_dest</i> — message reply to destination
Return Values	TRUE if the reply to destination was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetReplyTo fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>reply_to_dest</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_INVALID — <i>reply_to_dest</i> was not a valid subject
Description	TipcMsgSetReplyTo sets the reply to destination property of a message. The reply to destination property is a string representing a subject where a reply to the message should be sent.
Caution	TipcMsgSetReplyTo does not detect errors in <i>reply_to_dest</i> , such as misspelled subject names.
See Also	TipcMsgGetHeaderStrEncode
Examples	This example sets the reply to destination property of a message to <i>reply_subject</i>: <pre>if (!TipcMsgSetReplyTo(msg, "reply_subject")) { return; /* error */ }</pre>

TipcMsgSetSender

Name	TipcMsgSetSender — set the sender of a message
Synopsis	<pre>T_BOOL TipcMsgSetSender(msg, sender) T_IPC_MSG msg; T_STR sender;</pre>
Arguments	<i>msg</i> — message to set sender for <i>sender</i> — message sender
Return Values	TRUE if the sender was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetSender fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>sender</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_INVALID — <i>sender</i> was an empty string ("")
Description	TipcMsgSetSender sets the sender property of a message. The sender property is a string whose meaning depends on how the message is being used. Other applications of messages are free to use the sender and destination for another purpose. Message sender and destination properties can be null if they are not needed. TipcMsgSetSender makes a copy of the message sender. Messages sent with publish-subscribe between RTclients by RTserver, for example, use a subject name for the message sender and destination.
Caution	RTclients do not have to explicitly set the sender property of a message sent to RTserver; the functions TipcSrvMsgSend, TipcSrvMsgWrite, and TipcSrvMsgWriteVa all set the sender property of their messages to the unique subject of the process (which is stored in the option Unique_Subject).
See Also	TipcMsgGetSender, TipcMsgSetDest
Examples	This example sets the sender property of a message to _all: <pre>if (!TipcMsgSetSender(msg, "_all")) { return; /* error */ }</pre>

TipcMsgSetSenderTimestamp

Name	TipcMsgSetSenderTimestamp — set the sender timestamp property of a message
Synopsis	<pre>T_BOOL TipcMsgSetSenderTimestamp(msg, sender_timestamp) T_IPC_MSG msg; T_REAL8 sender_timestamp;</pre>
Arguments	<i>msg</i> — message to set sender timestamp for <i>sender_timestamp</i> — message sender timestamp
Return Values	TRUE if the sender timestamp was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetSenderTimestamp fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>sender_timestamp</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VAL_TOO_SMALL — <i>sender_timestamp</i> was negative
Description	TipcMsgSetSenderTimestamp sets the sender timestamp property of a message. A value of 0.0 indicates that the sender timestamp property was not set.
Caution	None
See Also	TipcMsgGetSenderTimestamp; see the <i>TIBCO SmartSockets Utilities</i> for information on TutGetWallTime.
Examples	This example sets the sender timestamp property of a message to the current time: <pre>if (!TipcMsgSetSenderTimestamp(msg, TutGetWallTime())) { return; /* error */ }</pre>

TipcMsgSetType

Name	TipcMsgSetType — set the type of a message
Synopsis	<pre>T_BOOL TipcMsgSetType(msg, mt) T_IPC_MSG msg; T_IPC_MT mt;</pre>
Arguments	<i>msg</i> — message to set type for <i>mt</i> — message type
Return Values	TRUE if the type was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetType fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>mt</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgSetType sets the type of a message. The type identifies the category of message being manipulated. Most connection callback types use a message type to allow various kinds of execution based on specific message types.
Caution	None
See Also	TipcMsgGetType
Examples	This example sets the type of a message to NUMERIC_DATA: <pre>mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA); if (mt == NULL) { return; /* error */ } if (!TipcMsgSetType(msg, mt)) { return; /* error */ }</pre>

TipcMsgSetUserProp

Name	TipcMsgSetUserProp — set the user-defined property of a message
Synopsis	<pre>T_BOOL TipcMsgSetUserProp(msg, user_prop) T_IPC_MSG msg; T_INT4 user_prop;</pre>
Arguments	<i>msg</i> — message to set user-defined property for <i>user_prop</i> — message user-defined property
Return Values	TRUE if the user-defined property was successfully set for the message, FALSE otherwise.
Diagnostics	If TipcMsgSetUserProp fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
Description	TipcMsgSetUserProp sets the user-defined property of a message. You can use the user-defined property for any purpose, such as attaching a version number to messages. TipcMsgSetUserProp overwrites the default value of the user-defined property set by TipcMtSetUserProp. See TipcMsgGetUserProp for more information on message user-defined properties.
Caution	None
See Also	TipcMsgGetUserProp
Examples	This example sets the user-defined property of a message to 2: <pre>if (!TipcMsgSetUserProp(msg, 2)) { return; /* error */ }</pre>

TpcMsgTraverse

Name	TpcMsgTraverse — traverse all fields in a message
Synopsis	<pre>T_PTR TpcMsgTraverse(msg, func, arg) T_IPC_MSG msg; T_IPC_MSG_TRAV_FUNC func; T_PTR arg;</pre>
Arguments	<i>msg</i> — message whose fields are traversed <i>func</i> — function to call once for each field <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	The first non-null return value from <i>func</i> , or NULL if <i>func</i> always returns NULL.
Diagnostics	If TpcMsgTraverse fails or traverses all the fields in <i>msg</i>, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or <i>func</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_END_OF_TRAVERSAL — all fields in <i>msg</i> were traversed
Description	TpcMsgTraverse traverses a message’s data fields in order and calls <i>func</i> once for each field. When <i>func</i> returns a non-null value, then TpcMsgTraverse returns with that value. One of the arguments passed to <i>func</i> is a T_IPC_MSG_TRAV pointer. This data structure contains a pointer to the field type (which is stored as a two-byte integer to save space), a pointer to the field value, and the array size. (Array size is set only if the field is an array type.)
Caution	If TpcMsgTraverse returns NULL, it may be because there was an error or simply because <i>func</i> always returned null.
See Also	TpcMsgPrint

Examples This example prints a message's field types:

```
T_PTR T_ENTRY print_one_field(msg, trav, arg)
T_IPC_MSG msg;
T_IPC_MSG_TRAV trav;
T_PTR arg; /* really (T_INT4 *) */
{
    T_INT4 *counter_ptr = arg;

    TutOut(" Field %d:\n", *counter_ptr);
    (*counter_ptr)++;
    TutOut(" Type: %d\n", (T_IPC_FT)(*trav->type_ptr));

    return NULL; /* continue traversal */
} /* print_one_field */

void print_field_offsets(msg)
T_IPC_MSG msg;
{
    T_INT4 num_fields;
    T_INT4 counter;

    if (!TipcMsgGetNumFields(msg, &num_fields)) {
        return; /* error */
    }
    TutOut("This message has %d fields.\n", num_fields);

    counter = 1;
    (void)TipcMsgTraverse(msg, print_one_field, &counter);
} /* print_field_offsets */
```


Chapter 10 **TipcMsgUpdateNamedBinary - TipcMsgWriteVa**

This chapter covers these TipcMsg API functions:

- *TipcMsgUpdateNamedBinary*
- *TipcMsgUpdateNamedBinaryPtr*
- *TipcMsgUpdateNamedBool*
- *TipcMsgUpdateNamedBoolArray*
- *TipcMsgUpdateNamedBoolArrayPtr*
- *TipcMsgUpdateNamedByte*
- *TipcMsgUpdateNamedChar*
- *TipcMsgUpdateNamedInt2*
- *TipcMsgUpdateNamedInt2Array*
- *TipcMsgUpdateNamedInt2ArrayPtr*
- *TipcMsgUpdateNamedInt4*
- *TipcMsgUpdateNamedInt4Array*
- *TipcMsgUpdateNamedInt4ArrayPtr*
- *TipcMsgUpdateNamedInt8*
- *TipcMsgUpdateNamedInt8Array*
- *TipcMsgUpdateNamedInt8ArrayPtr*
- *TipcMsgUpdateNamedMsg*
- *TipcMsgUpdateNamedMsgArray*
- *TipcMsgUpdateNamedMsgArrayPtr*
- *TipcMsgUpdateNamedMsgPtr*
- *TipcMsgUpdateNamedReal4*
- *TipcMsgUpdateNamedReal4Array*
- *TipcMsgUpdateNamedReal4ArrayPtr*
- *TipcMsgUpdateNamedReal8*

- *TpcMsgUpdateNamedReal8Array*
- *TpcMsgUpdateNamedReal8ArrayPtr*
- *TpcMsgUpdateNamedReal16*
- *TpcMsgUpdateNamedReal16Array*
- *TpcMsgUpdateNamedReal16ArrayPtr*
- *TpcMsgUpdateNamedStr*
- *TpcMsgUpdateNamedStrArray*
- *TpcMsgUpdateNamedStrArrayPtr*
- *TpcMsgUpdateNamedStrPtr*
- *TpcMsgUpdateNamedTimestamp*
- *TpcMsgUpdateNamedTimestampArray*
- *TpcMsgUpdateNamedTimestampArrayPtr*
- *TpcMsgUpdateNamedUnknown*
- *TpcMsgUpdateNamedUtf8*
- *TpcMsgUpdateNamedUtf8Array*
- *TpcMsgUpdateNamedUtf8ArrayPtr*
- *TpcMsgUpdateNamedUtf8Ptr*
- *TpcMsgUpdateNamedXml*
- *TpcMsgUpdateNamedXmlPtr*
- *TpcMsgWrite*
- *TpcMsgWriteVa*

TipcMsgUpdateNamedBinary

Name	TipcMsgUpdateNamedBinary — update a named BINARY field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedBinary(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_PTR value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedBinary fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedBinary updates a BINARY field in a message, associating that field with a name.
See Also	TipcMsgAddNamedBinary, TipcMsgGetNamedBinary
Examples	This example changes the value of the named field "data": <pre>T_PTR data; T_INT4 size; if (!TipcMsgUpdateNamedBinary(msg, "data", data, size)) { return; /* error */ }</pre>

TipcMsgUpdateNamedBinaryPtr

Name	TipcMsgUpdateNamedBinaryPtr — update a pointer to the value of a BINARY field with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedBinaryPtr(msg, name, ptr, size, field_return) T_IPC_MSG msg; T_STR name; T_PTR ptr; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedBinaryPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedBinaryPtr updates a BINARY field in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAddNamedBinaryPtr

Examples This example changes the value of the named field "data":

```
T_PTR data;
T_INT4 size;
if (!TipcMsgUpdateNamedBinaryPtr(msg, "data", data, size, T_NULL))
{
    return; /* error */
}
```

TipcMsgUpdateNamedBool

Name	TipcMsgUpdateNamedBool — update a named BOOL field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedBool(msg, name, value) T_IPC_MSG msg; T_STR name; T_BOOL value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedBool fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedBool updates a BOOL field specified by field name.
Caution	None
See Also	TipcMsgAddNamedBool, TipcMsgGetNamedBool
Examples	This example changes the value of the named field "data": <pre>T_BOOL data; if (!TipcMsgUpdateNamedBool(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedBoolArray

Name	TipcMsgUpdateNamedBoolArray — update a named BOOL_ARRAY field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedBoolArray(msg, name, *value, size) T_IPC_MSG msg; T_STR name; T_BOOL *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedBoolArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedBoolArray updates a BOOL_ARRAY field specified by field name.
Caution	None
See Also	TipcMsgAddNamedBoolArray, TipcMsgGetNamedBoolArray
Examples	This example changes the value of the named field "data": <pre>T_BOOL *data; T_INT4 size; if (!TipcMsgUpdateNamedBoolArray(msg, "data", data, size)) { return; /* error */ }</pre>

TipcMsgUpdateNamedBoolArrayPtr

Name	TipcMsgUpdateNamedBoolArrayPtr — update a pointer to a named BOOL_ARRAY field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedBoolArrayPtr(msg, name, *value, size, *field_return) T_IPC_MSG msg; T_STR name; T_BOOL *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedBoolArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedBoolArrayPtr updates a BOOL_ARRAY pointer field specified by field name.
Caution	None
See Also	TipcMsgAddNamedBoolArrayPtr

Examples This example changes the value of the named field "data":

```
T_BOOL *data;
T_INT4 size;
if (!TipcMsgUpdateNamedBoolArrayPtr(msg, "data", data, size,
T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedByte

Name	TipcMsgUpdateNamedByte — update a named BYTE field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedByte(msg, name, value) T_IPC_MSG msg; T_STR name; T_UCHAR value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedByte fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedByte updates a BYTE field specified by field name.
Caution	None
See Also	TipcMsgAddNamedByte, TipcMsgUpdateNamedByte
Examples	This example changes the value of the named field "data": <pre>T_UCHAR data; if (!TipcMsgUpdateNamedByte(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedChar

Name	TipcMsgUpdateNamedChar — update a CHAR field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedChar(msg, name, value) T_IPC_MSG msg; T_STR name; T_CHAR value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedChar fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedChar updates a character field in a message, associating that field with a name.
See Also	TipcMsgAddNamedChar, TipcMsgGetNamedChar
Examples	This example changes the value of the named field "data": <pre>T_CHAR data; if (!TipcMsgUpdateNamedChar(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedInt2

Name	TipcMsgUpdateNamedInt2 — update an INT2 field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedInt2(msg, name, value) T_IPC_MSG msg; T_STR name; T_INT2 value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedInt2 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedInt2 updates an INT2 field in a message, associating that field with a name.
See Also	TipcMsgAddNamedInt2, TipcMsgGetNamedInt2
Examples	This example changes the value of the named field "data": <pre>T_INT2 data; if (!TipcMsgUpdateNamedInt2(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedInt2Array

Name	TipcMsgUpdateNamedInt2Array — update a field containing an array of INT2 fields in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedInt2Array(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_INT2 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedInt2Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedInt2Array updates a field containing an array of INT2 fields in a message, associating that field with a name.
See Also	TipcMsgAddNamedInt2Array, TipcMsgGetNamedInt2Array
Examples	This example changes the value of the named field "data": <pre>T_INT2 *data; T_INT4 size; if (!TipcMsgUpdateNamedInt2Array(msg, "data", data, size)) { return; /* error */ }</pre>

TipcMsgUpdateNamedInt2ArrayPtr

Name	TipcMsgUpdateNamedInt2ArrayPtr — update a pointer to the value of a field containing an array of INT2 fields
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedInt2ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_INT2 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedInt2ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedInt2ArrayPtr updates a field containing an array of INT2 fields in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAddNamedInt2ArrayPtr

Examples This example changes the value of the named field "data":

```
T_INT2 *data;
T_INT4 size;
if (!TipcMsgUpdateNamedInt2ArrayPtr(msg, "data", data, size,
T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedInt4

Name	TipcMsgUpdateNamedInt4 — update an INT4 field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedInt4(msg, name, value) T_IPC_MSG msg; T_STR name; T_INT4 value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedInt4 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedInt4 updates an INT4 field in a message, associating that field with a name.
See Also	TipcMsgAddNamedInt4, TipcMsgGetNamedInt4
Examples	This example changes the value of the named field "data": <pre>T_INT4 data; if (!TipcMsgUpdateNamedInt4(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedInt4Array

Name	TipcMsgUpdateNamedInt4Array — update a field containing an array of INT4 fields in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedInt4Array(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_INT4 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedInt4Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedInt4Array updates a field containing an array of INT4 fields in a message, associating that field with a name.
See Also	TipcMsgAddNamedInt4Array, TipcMsgGetNamedInt4Array
Examples	This example changes the value of the named field "data": <pre>T_INT4 *data; T_INT4 size; if (!TipcMsgUpdateNamedInt4Array(msg, "data", data, size)) { return; /* error */ }</pre>

TipcMsgUpdateNamedInt4ArrayPtr

Name	TipcMsgUpdateNamedInt4ArrayPtr — update a pointer to the value of a named field containing an array of INT4 fields
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedInt4ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_INT4 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedInt4ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedInt4ArrayPtr updates a field containing an array of INT4 fields in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAddNamedInt4ArrayPtr

Examples This example changes the value of the named field "data":

```
T_INT4 *data;
T_INT4 size;
if (!TipcMsgUpdateNamedInt4ArrayPtr(msg, "data", data, size,
T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedInt8

Name	TipcMsgUpdateNamedInt8 — update an INT8 field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedInt8(msg, name, value) T_IPC_MSG msg; T_STR name; T_INT8 value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedInt8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedInt8 updates an INT8 field in a message, associating that field with a name.
See Also	TipcMsgAddNamedInt8, TipcMsgGetNamedInt8
Examples	This example changes the value of the named field "data": <pre>T_INT8 data; if (!TipcMsgUpdateNamedInt8(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedInt8Array

Name	TipcMsgUpdateNamedInt8Array — update a field containing an array of INT8 fields in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedInt8Array(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_INT8 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedInt8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedInt8Array updates a field containing an array of INT8 fields in a message, associating that field with a name.
See Also	TipcMsgAddNamedInt8Array, TipcMsgGetNamedInt8Array
Examples	This example changes the value of the named field "data": <pre>T_INT8 *data; T_INT4 size; if (!TipcMsgUpdateNamedInt8Array(msg, "data", data, size)) { return; /* error */ }</pre>

TipcMsgUpdateNamedInt8ArrayPtr

Name	TipcMsgUpdateNamedInt8ArrayPtr — update a pointer to the value of a named field containing an array of INT8 fields
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedInt8ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_INT8 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedInt8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedInt8ArrayPtr updates a field containing an array of INT8 fields in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAddNamedInt8ArrayPtr

Examples This example changes the value of the named field "data":

```
T_INT8 *data;
T_INT4 size;
if (!TipcMsgUpdateNamedInt8ArrayPtr(msg, "data", data, size,
T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedMsg

Name	TipcMsgUpdateNamedMsg — update a MSG field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedMsg(msg, name, value) T_IPC_MSG msg; T_STR name; T_IPC_MSG value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedMsg fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedMsg updates a MSG field in a message, associating that field with a name.
See Also	TipcMsgAddNamedMsg, TipcMsgGetNamedMsg
Examples	This example changes the value of the named field "data": <pre>T_IPC_MSG data; if (!TipcMsgUpdateNamedMsg(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedMsgArray

Name	TipcMsgUpdateNamedMsgArray — update a field containing an array of MSG fields in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedMsgArray(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_MSG *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedMsgArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedMsgArray updates a field containing an array of MSG fields in a message, associating that field with a name.
See Also	TipcMsgAddNamedMsgArray, TipcMsgGetNamedMsgArray
Examples	This example changes the value of the named field "data": <pre>T_IPC_MSG *data; T_INT4 size; if (!TipcMsgUpdateNamedMsgArray(msg, "data", data, size)) { return; /* error */ }</pre>

TpcMsgUpdateNamedMsgArrayPtr

Name	TpcMsgUpdateNamedMsgArrayPtr — update a pointer to the value of a named field containing an array of MSG fields
Synopsis	<pre>T_BOOL TpcMsgUpdateNamedMsgArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_MSG *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TpcMsgUpdateNamedMsgArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: - T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null - T_ERR_MSG_INVALID — <i>msg</i> was not a valid message - T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message - T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. - T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TpcMsgUpdateNamedMsgArrayPtr updates a field containing an array of MSG fields in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TpcMsgAddNamedMsgArrayPtr

Examples This example changes the value of the named field "data":

```
T_IPC_MSG *data;
T_INT4 size;
if (!TipcMsgUpdateNamedMsgArrayPtr(msg, "data", data, size,
T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedMsgPtr

Name	TipcMsgUpdateNamedMsgPtr — update a named MSG pointer field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedMsgPtr(msg, name, value, *field_return) T_IPC_MSG msg; T_STR name; T_MSG *value; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedMsgPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedMsgPtr updates a MSG pointer field specified by field name.
See Also	TipcMsgAddNamedMsgPtr
Examples	This example changes the value of the named field "data": <pre>T_IPC_MSG data; if (!TipcMsgUpdateNamedMsgPtr(msg, "data", data, T_NULL)) { return; /* error */ }</pre>

TipcMsgUpdateNamedReal4

Name	TipcMsgUpdateNamedReal4 — update a REAL4 field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedReal4(msg, name, value) T_IPC_MSG msg; T_STR name; T_REAL4 value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedReal4 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedReal4 updates a REAL4 field in a message, associating that field with a name.
See Also	TipcMsgAddNamedReal4, TipcMsgGetNamedReal4
Examples	This example changes the value of the named field "data": <pre>T_REAL4 data; if (!TipcMsgUpdateNamedReal4(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedReal4Array

Name	TipcMsgUpdateNamedReal4Array — update a field containing an array of REAL4 fields in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedReal4Array(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_REAL4 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedReal4Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedReal4Array updates a field containing an array of REAL4 fields in a message, associating that field with a name.
See Also	TipcMsgAddNamedReal4Array, TipcMsgGetNamedReal4Array
Examples	This example changes the value of the named field "data": <pre>T_REAL4 *data; T_INT4 size; if (!TipcMsgUpdateNamedReal4Array(msg, "data", data, size)) { return; /* error */ }</pre>

TipcMsgUpdateNamedReal4ArrayPtr

Name	TipcMsgUpdateNamedReal4ArrayPtr — update a pointer to the value of a named field containing an array of REAL4 fields
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedReal4ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_REAL4 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedReal4ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedReal4ArrayPtr updates a field containing an array of REAL4 fields in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAddNamedReal4ArrayPtr

Examples This example changes the value of the named field "data":

```
T_REAL4 *data;
T_INT4 size;
if (!TipcMsgUpdateNamedReal4ArrayPtr(msg, "data", data, size,
T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedReal8

Name	TipcMsgUpdateNamedReal8 — update a REAL8 field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedReal8(msg, name, value) T_IPC_MSG msg; T_STR name; T_REAL8 value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedReal8 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedReal8 updates a REAL8 field in a message, associating that field with a name.
See Also	TipcMsgAddNamedReal8, TipcMsgGetNamedReal8
Examples	This example changes the value of the named field "data": <pre>T_REAL8 data; if (!TipcMsgUpdateNamedReal8(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedReal8Array

Name	TipcMsgUpdateNamedReal8Array — update a field containing an array of REAL8 fields in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedReal8Array(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_REAL8 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedReal8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedReal8Array updates a field containing an array of REAL8 fields in a message, associating that field with a name.
See Also	TipcMsgAddNamedReal8Array, TipcMsgGetNamedReal8Array
Examples	This example changes the value of the named field "data": <pre>T_REAL8 *data; T_INT4 size; if (!TipcMsgUpdateNamedReal8Array(msg, "data", data, size)) { return; /* error */ }</pre>

TipcMsgUpdateNamedReal8ArrayPtr

Name	TipcMsgUpdateNamedReal8ArrayPtr — update a pointer to the value of a named field containing an array of REAL8 fields
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedReal8ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_REAL8 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedReal8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedReal8ArrayPtr updates a field containing an array of REAL8 fields in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAddNamedReal8ArrayPtr

Examples This example changes the value of the named field "data":

```
T_REAL8 *data;
T_INT4 size;
if (!TipcMsgUpdateNamedReal8ArrayPtr(msg, "data", data, size,
T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedReal16

Name	TipcMsgUpdateNamedReal16 — update a REAL16 field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedReal16(msg, name, value) T_IPC_MSG msg; T_STR name; T_REAL16 value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedReal16 fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedReal16 updates a REAL16 field in a message, associating that field with a name.
	Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.
See Also	TipcMsgAddNamedReal16, TipcMsgGetNamedReal16
Examples	This example changes the value of the named field "data": <pre>T_REAL16 data; if (!TipcMsgUpdateNamedReal16(msg, "data", data)) { return; /* error */ }</pre>

TpcMsgUpdateNamedReal16Array

Name TpcMsgUpdateNamedReal16Array — update a field containing an array of REAL16 fields in a message with new data

Synopsis

```
T_BOOL TpcMsgUpdateNamedReal16Array(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_REAL16 *value;
T_INT4 size;
```

Arguments *msg* — message to update field in
name — name of the field
value — value of the field
size — number of elements in the field

Return Values TRUE if the update was successful, FALSE otherwise.

Diagnostics If TpcMsgUpdateNamedReal16Array fails, it returns FALSE and sets the global SmartSockets error number to one of:

- T_ERR_NULL_PTR — *msg*, *name* or *value* is null
- T_ERR_MSG_INVALID — *msg* was not a valid message
- T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message
- T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message.
- T_ERR_VALUE_INVALID — *value* was not a valid value

Description TpcMsgUpdateNamedReal16Array updates a field containing an array of REAL16 fields in a message, associating that field with a name.



Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.

See Also TpcMsgAddNamedReal16Array, TpcMsgGetNamedReal16Array

Examples This example changes the value of the named field "data":

```
T_REAL16 *data;
T_INT4 size;
if (!TipcMsgUpdateNamedReal16Array(msg, "data", data, size)) {
    return; /* error */
}
```

TipcMsgUpdateNamedReal16ArrayPtr

Name	TipcMsgUpdateNamedReal16ArrayPtr — update a pointer to the value of a named field containing an array of REAL16 fields
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedReal16ArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_REAL16 *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedReal16ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedReal16ArrayPtr updates a field containing an array of REAL16 fields in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.



Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.

See Also `TipcMsgAddNamedReal16ArrayPtr`

Examples This example changes the value of the named field "data":

```
T_REAL16 *data;
T_INT4 size;
if (!TipcMsgUpdateNamedReal16ArrayPtr(msg, "data", data, size,
T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedStr

Name	TipcMsgUpdateNamedStr — update a string field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedStr(msg, name, value) T_IPC_MSG msg; T_STR name; T_STR value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedStr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedStr updates a string field in a message, associating that field with a name.
See Also	TipcMsgAddNamedStr, TipcMsgGetNamedStr
Examples	This example changes the value of the named field "data": <pre>T_STR data; if (!TipcMsgUpdateNamedStr(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedStrArray

Name	TipcMsgUpdateNamedStrArray — update a field containing an array of STR fields in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedStrArray(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_STR *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedStrArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedStrArray appends a field containing an array of STR fields in a message, associating that field with a name.
See Also	TipcMsgAddNamedStrArray, TipcMsgGetNamedStrArray
Examples	This example changes the value of the named field "data": <pre>T_STR *data; T_INT4 size; if (!TipcMsgUpdateNamedStrArray(msg, "data", data, size)) { return; /* error */ }</pre>

TipcMsgUpdateNamedStrArrayPtr

Name	TipcMsgUpdateNamedStrArrayPtr — update a pointer to the value of a named field containing an array of STR fields
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedStrArrayPtr(msg, name, value, size, field_return) T_IPC_MSG msg; T_STR name; T_STR *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the append was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedStrArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedStrArrayPtr appends a field containing an array of STR fields in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAddNamedStrArrayPtr

Examples This example changes the value of the named field "data":

```
T_STR *data;
T_INT4 size;
if (!TipcMsgUpdateNamedStrArrayPtr(msg, "data", data, size,
T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedStrPtr

Name	TipcMsgUpdateNamedStrPtr — update a pointer to the value of a named STR field in a message
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedStrPtr(msg, name, ptr, field_return) T_IPC_MSG msg; T_STR name; T_PTR ptr; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to append field to <i>name</i> — name of the field <i>ptr</i> — pointer to the value of the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the append was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedStrPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedStrPtr appends a string field in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAddNamedStrPtr
Examples	This example changes the value of the named field "data": <pre>T_STR data; if (!TipcMsgUpdateNamedStrPtr(msg, "data", data, T_NULL)) { return; /* error */ }</pre>

TipcMsgUpdateNamedTimestamp

Name	TipcMsgUpdateNamedTimestamp — update a <code>TIMESTAMP</code> field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedTimestamp(msg, name, value) T_IPC_MSG msg; T_STR name; T_REAL8 value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedTimestamp fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_NULL_PTR</code> — <i>msg</i>, <i>name</i> or <i>value</i> is null • <code>T_ERR_MSG_INVALID</code> — <i>msg</i> was not a valid message • <code>T_ERR_FT_MISMATCH</code> — the message type grammar did not match the fields in the message • <code>T_ERR_READ_ONLY</code> — the message was marked read-only. This happens if the message is a field in another message. • <code>T_ERR_VALUE_INVALID</code> — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedTimestamp updates a <code>TIMESTAMP</code> field in a message, associating that field with a name.
See Also	TipcMsgAddNamedTimestamp, TipcMsgGetNamedTimestamp
Examples	This example changes the value of the named field "data": <pre>T_REAL8 data; if (!TipcMsgUpdateNamedTimestamp(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedTimestampArray

Name	TipcMsgUpdateNamedTimestampArray — update a field containing an array of TIMESTAMP fields in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedTimestampArray(msg, name, value, size) T_IPC_MSG msg; T_STR name; T_REAL8 *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedTimestampArray fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedTimestampArray updates a field containing an array of TIMESTAMP fields in a message, associating that field with a name.
See Also	TipcMsgAddNamedTimestampArray, TipcMsgGetNamedTimestampArray
Examples	This example changes the value of the named field "data": <pre>T_REAL8 *data; T_INT4 size; if (!TipcMsgUpdateNamedTimestampArray(msg, "data", data, size)) { return; /* error */ }</pre>

TipcMsgUpdateNamedTimestampArrayPtr

Name	TipcMsgUpdateNamedTimestampArrayPtr — update a pointer to the value of a named field containing an array of <code>TIMESTAMP</code> fields
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedTimestampArrayPtr(<i>msg</i>, <i>name</i>, <i>value</i>, <i>size</i>, <i>field_return</i>) T_IPC_MSG <i>msg</i>; T_STR <i>name</i>; T_REAL8 *<i>value</i>; T_INT4 <i>size</i>; T_IPC_MSG_FIELD *<i>field_return</i>;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use <code>NULL</code> if not needed)
Return Values	<code>TRUE</code> if the update was successful, <code>FALSE</code> otherwise.
Diagnostics	If <code>TipcMsgUpdateNamedTimestampArrayPtr</code> fails, it returns <code>FALSE</code> and sets the global SmartSockets error number to one of: <code>T_ERR_NULL_PTR</code> — <i>msg</i>, <i>name</i> or <i>value</i> is null <code>T_ERR_MSG_INVALID</code> — <i>msg</i> was not a valid message <code>T_ERR_FT_MISMATCH</code> — the message type grammar did not match the fields in the message <code>T_ERR_READ_ONLY</code> — the message was marked read-only. This happens if the message is a field in another message. <code>T_ERR_VALUE_INVALID</code> — <i>value</i> was not a valid value
Description	<code>TipcMsgUpdateNamedTimestampArrayPtr</code> updates a field containing an array of <code>TIMESTAMP</code> fields in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	<code>TipcMsgAddNamedTimestampArrayPtr</code>

Examples This example changes the value of the named field "data":

```
T_REAL8 *data;
T_INT4 size;
if (!TipcMsgUpdateNamedTimestampArrayPtr(msg, "data", data, size,
                                           T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedUnknown

Name	TipcMsgUpdateNamedUnknown — update a field with an unknown or non-existent value in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedUnknown(msg, name, type) T_IPC_MSG msg; T_STR name; T_IPC_FT type;</pre>
Arguments	<i>msg</i> — message to append field to <i>name</i> — name of the field <i>type</i> — type of the field (such as T_IPC_FT_INT2, T_IPC_FT_INT4)
Return Values	TRUE if the append was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedUnknown fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedUnknown updates a named field with an unknown or non-existent value.
See Also	TipcMsgAddNamedUnknown, TipcMsgGetNamedUnknown
Examples	This example changes the value of the named field "data": <pre>T_STR data; if (!TipcMsgUpdateNamedUnknown(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedUtf8

Name	TipcMsgUpdateNamedUtf8 — update a named UTF8 field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedUtf8(msg, name, value) T_IPC_MSG msg; T_STR name; T_STR value;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedUtf8 fails, it returns FALSE and sets the global SmartSockets error number to one of: T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null T_ERR_MSG_INVALID — <i>msg</i> was not a valid message T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message T_ERR_READ_ONLY — <i>msg</i> was marked read-only. This happens if the message is a field in another message. T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedUtf8 updates a UTF8 field specified by field name.
See Also	TipcMsgAddNamedUtf8, TipcMsgGetNamedUtf8
Examples	This example changes the value of the named field "data": <pre>T_STR data; if (!TipcMsgUpdateNamedUtf8(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedUtf8Array

Name	TipcMsgUpdateNamedUtf8Array — update a named UTF8_ARRAY field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedUtf8Array(msg, name, *value, size) T_IPC_MSG msg; T_STR name; T_STR *value; T_INT4 size;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field <i>size</i> — number of elements in the field
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedUtf8Array fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — <i>msg</i> was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedUtf8Array updates a UTF8_ARRAY field specified by field name.
See Also	TipcMsgAddNamedUtf8Array, TipcMsgGetNamedUtf8Array
Examples	This example changes the value of the named field "data": <pre>T_STR *data; T_INT4 size; if (!TipcMsgUpdateNamedUtf8Array(msg, "data", data, size)) { return; /* error */ }</pre>

TipcMsgUpdateNamedUtf8ArrayPtr

Name	TipcMsgUpdateNamedUtf8ArrayPtr — update a named pointer to a field containing an array of UTF8 fields in a message
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedUtf8ArrayPtr(msg, name, *value, size, *field_return) T_IPC_MSG msg; T_STR name; T_STR *value; T_INT4 size; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field <i>size</i> — number of elements in the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedUtf8ArrayPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — <i>msg</i> was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedUtf8ArrayPtr updates a UTF8_ARRAY pointer field specified by field name.
See Also	TipcMsgAddNamedUtf8ArrayPtr

Examples This example changes the value of the named field "data":

```
T_STR *data;
T_INT4 size;
if (!TipcMsgUpdateNamedUtf8ArrayPtr(msg, "data", data, size,
T_NULL)) {
    return; /* error */
}
```

TipcMsgUpdateNamedUtf8Ptr

Name	TipcMsgUpdateNamedUtf8Ptr — update a named UTF8 pointer field in a message with new data
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedUtf8Ptr(msg, name, value, *field_return) T_IPC_MSG msg; T_STR name; T_STR value; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to update field in <i>name</i> — name of the field <i>value</i> — new value of the field <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the update was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedUtf8Ptr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — <i>msg</i> was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedUtf8Ptr updates a UTF8 pointer field specified by field name.
See Also	TipcMsgAddNamedUtf8Ptr
Examples	This example changes the value of the named field "data": <pre>T_STR data; T_STR foo; if (!TipcMsgUpdateNamedUtf8Ptr(msg, "data", "foo", field_return)) return; /* error */ }</pre>

TipcMsgUpdateNamedXml

Name	TipcMsgUpdateNamedXml — update a named XML field in a message
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedXml(msg, name, value) T_IPC_MSG msg; T_STR name; T_XML value;</pre>
Arguments	<i>msg</i> — message to which to append the field <i>name</i> — name of the field <i>value</i> — value of the field
Return Values	TRUE if the append was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedXml fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedXml appends an XML object in a message, associating that field with a name.
See Also	TipcMsgAddNamedXml, TipcMsgGetNamedXml
Examples	This example changes the value of the named field "data": <pre>T_XML data; if (!TipcMsgUpdateNamedXml(msg, "data", data)) { return; /* error */ }</pre>

TipcMsgUpdateNamedXmlPtr

Name	TipcMsgUpdateNamedXmlPtr — update a pointer to the value of a named XML field in a message
Synopsis	<pre>T_BOOL TipcMsgUpdateNamedXmlPtr(msg, name, value, field_return) T_IPC_MSG msg; T_STR name; T_XML value; T_IPC_MSG_FIELD *field_return;</pre>
Arguments	<i>msg</i> — message to which to append the field <i>name</i> — name of the field <i>value</i> — XML object to append to message <i>field_return</i> — optional value to return field (use NULL if not needed)
Return Values	TRUE if the append was successful, FALSE otherwise.
Diagnostics	If TipcMsgUpdateNamedXmlPtr fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i>, <i>name</i> or <i>value</i> is null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_MISMATCH — the message type grammar did not match the fields in the message • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • T_ERR_VALUE_INVALID — <i>value</i> was not a valid value
Description	TipcMsgUpdateNamedXmlPtr appends an XML object in a message, associating that field with a name. The value given is used as a pointer to the value, so the copy is made when the message is actually sent, not when the call is made. You must ensure that the data pointed to remains valid until the message is sent.
See Also	TipcMsgAddNamedXmlPtr
Examples	This example changes the value of the named field "data": <pre>T_XML data; if (!TipcMsgUpdateNamedXmlPtr(msg, "data", data, T_NULL)) { return; /* error */ }</pre>

TipcMsgWrite

Name	TipcMsgWrite — append fields to a message
Synopsis	<pre>T_BOOL TipcMsgWrite(msg, ...) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to append fields to additional arguments — null-terminated field type-value pairs
Return Values	TRUE if all the fields were successfully appended, FALSE otherwise.
Diagnostics	If TipcMsgWrite fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_READ_ONLY — the message was marked read-only. This happens if the message is a field in another message. • any error number from TipcMsgWriteVa
Description	TipcMsgWrite is a convenience function that takes a null-terminated variable number of field type-value pairs and appends these fields to the message. The field type part of the pair is a T_IPC_FT, and the value part of the pair corresponds to the argument(s) for the related TipcMsgAppend* function (for example, if the type part is T_IPC_FT_REAL8, then the value part is a REAL8 number) or TipcMsgSet* function (for example, if the type part is T_IPC_PROP_DELIVERY_MODE, then the value part is a T_IPC_DELIVERY_MODE setting). TipcMsgWrite is a short wrapper function that calls TipcMsgWriteVa.
Caution	The last argument passed to TipcMsgWrite must be null; otherwise TipcMsgWrite may append garbage fields to the message. Using TipcMsgWrite is slightly slower than using the TipcMsgAppend* functions directly. TipcMsgWrite does not send a message: it only appends fields to a message.
See Also	TipcMsgRead, TipcMsgWriteVa

Examples This example creates a message, appends two fields with `TipcMsgWrite`, and destroys the message:

```
T_IPC_MT mt;
T_IPC_MSG msg;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

/* An RTclient would typically set the destination of */
/* a message at this point by calling TipcMsgSetDest. */

if (!TipcMsgWrite(msg,
                  T_IPC_FT_STR, "speed_limit",
                  T_IPC_FT_REAL8, 55.0,
                  NULL)) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcMsgWriteVa

Name	TipcMsgWriteVa — append fields to a message (<i>va_list</i> version)
Synopsis	<pre>T_BOOL TipcMsgWriteVa(msg, var_arg_list) T_IPC_MSG msg; va_list var_arg_list;</pre>
Arguments	<i>msg</i> — message to append fields to <i>var_arg_list</i> — argument list
Return Values	TRUE if all the fields were successfully appended, FALSE otherwise.
Diagnostics	If TipcMsgWriteVa fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_FT_INVALID — a field type argument was not a valid field type • any error number from TipcMsgAppend*
Description	TipcMsgWriteVa is the non-varargs version of TipcMsgWrite (TipcMsgWrite actually calls TipcMsgWriteVa). TipcMsgWriteVa can be used to implement a varargs function. In the C language, a varargs function (one that takes a variable number of arguments) cannot call another varargs function with the original arguments, due to the way that C implements varargs. A varargs function can (and usually does) call a helper function that takes a fixed number of arguments, with the last argument usually of type <i>va_list</i>. TipcMsgWriteVa is TipcMsgWrite's helper function.
Caution	TipcMsgWriteVa does not send a message: it only appends fields to a message. TipcMsgWriteVa does not support fields with a value of unknown.
See Also	TipcMsgRead, TipcMsgUpdateNamedBinary, TipcMsgAppend*

Examples This example shows how to write a varargs function in ANSI C that creates a message and appends fields to it:

```
T_IPC_MSG T_ENTRY my_msg_create_write(T_INT4 mt_num, ...)
{
    T_IPC_MT mt;
    T_IPC_MSG msg;
    va_list var_arg_list;
    T_BOOL status;

    /* look up the message type */
    mt = TipcMtLookupByNum(mt_num);
    if (mt == NULL) {
        return; /* error */
    }

    /* create the message */
    msg = TipcMsgCreate(mt);
    if (msg == NULL) {
        return; /* error */
    }

    /* An RTclient would typically set the destination of */
    /* a message at this point by calling TipcMsgSetDest. */

    /* retrieve the varargs and use them */
    va_start(var_arg_list, mt_num);
    status = TipcMsgWriteVa(msg, var_arg_list);
    va_end(var_arg_list);
    if (!status) {
        if (!TipcMsgDestroy(msg)) {
            return; /* error */
        }
        msg = NULL;
    }

    return msg;
} /* my_msg_create_write */
```



```

/* ===== */
/*...code from calling function below */

/* fragment that uses my_msg_create_write */
T_IPC_MSG msg;

msg = my_msg_create_write(T_MT_NUMERIC_DATA,
                          T_IPC_PROP_PRIORITY, 2,
                          T_IPC_FT_STR, "voltage",
                          T_IPC_FT_REAL8, 120.0,
                          T_IPC_FT_STR, "temperature",
                          T_IPC_FT_REAL8, 98.6,
                          NULL);

if (msg == NULL) {
    return; /* error */
}

/* A process would typically send a message at this point by */
/* calling TipcConnMsgSend or TipcSrvMsgSend. */

/* A process would typically destroy a message at this point by */
/* calling TipcMsgDestroy. */

```


Chapter 11 **TipcMt***

This chapter covers the TipcMt API functions:

- *TipcMtCreate*
- *TipcMtDestroy*
- *TipcMtGetCompression*
- *TipcMtGetDeliveryMode*
- *TipcMtGetDeliveryMode*
- *TipcMtGetDeliveryTimeout*
- *TipcMtGetGrammar*
- *TipcMtGetHeaderStrEncode*
- *TipcMtGetLbMode*
- *TipcMtGetName*
- *TipcMtGetNum*
- *TipcMtGetPriority*
- *TipcMtGetUserProp*
- *TipcMtLogAddMt*
- *TipcMtLogRemoveMt*
- *TipcMtLookup*
- *TipcMtLookupByNum*
- *TipcMtPrint*
- *TipcMtSetCompression*
- *TipcMtSetDeliveryMode*
- *TipcMtSetDeliveryTimeout*
- *TipcMtSetHeaderStrEncode*
- *TipcMtSetLbMode*
- *TipcMtSetPriority*
- *TipcMtSetPriorityUnknown*

- *TipcMtSetUserProp*
- *TipcMtTraverse*

TipcMtCreate

Name	TipcMtCreate — create a new message type
Synopsis	<pre>T_IPC_MT TipcMtCreate(name, num, grammar) T_STR name; T_INT4 num; T_STR grammar;</pre>
Arguments	<i>name</i> — name of new message type (not case-sensitive) <i>num</i> — number of new message type <i>grammar</i> — description for reading and writing message files
Return Values	New message type if successful, NULL otherwise.
Diagnostics	If TipcMtCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>name</i> was null or <i>grammar</i> was null • T_ERR_ALREADY_EXISTS — a message type with the desired <i>name</i> or <i>num</i> already exists • T_ERR_VAL_INVALID — <i>grammar</i> was not valid
Description	TipcMtCreate creates a new message type. A message type is a template for a specific kind of message. Once the message type is created, any number of messages of that type can be created. The message type <i>name</i> should be a legal identifier. The message type number <i>num</i> is a signed four-byte integer. Message type numbers less than 1 are reserved for SmartSockets standard message types. The standard SmartSockets message types use similar names and numbers. (For example, the message type with name "numeric_data" has a C #define defined number T_MT_NUMERIC_DATA.) See the <i>TIBCO SmartSockets User's Guide</i> for a list of all standard message types. The message type <i>grammar</i> identifies the layout of fields in messages that use this message type. The grammar consists of a list of field types. See the <i>TIBCO SmartSockets User's Guide</i> for more information on message type grammars.

The new message type is created with an unset priority property, a delivery mode property of `T_IPC_DELIVERY_BEST_EFFORT`, an unset delivery timeout property, a load balancing mode property of `T_IPC_LB_NONE`, a header string encode property of `FALSE`, and a user-defined property of 0. These properties can be set with the functions `TipcMtSetPriority`, `TipcMtSetDeliveryMode`, `TipcMtSetDeliveryTimeout`, `TipcMtSetLbMode`, `TipcMtSetHeaderStrEncode`, and `TipcMtSetUserProp`, respectively.

`TipcMtCreate` makes a copy of *name* and *grammar*.

Caution User-defined message types must be created in all processes that will use the message types. Because there is no central SmartSockets message type repository, the recommended approach is to write an initialization function that calls `TipcMtCreate` as needed, link that function into all user-defined programs, and then call the function during program initialization.

See Also `TipcMtDestroy`, `TipcMtLookup`, `TipcMtLookupByNum`

Examples This example creates a message type named `XYZ_COORD` which uses three integer coordinates, creates an `XYZ_COORD` message, appends fields, and writes the message to a message file:

```
#define USER_MT_XYZ_COORD 2000

T_IPC_MT mt;
T_IPC_MSG msg;
T_IPC_MSG_FILE msg_file;

/* create the message type */
mt = TipcMtCreate("xyz_coord", USER_MT_XYZ_COORD, "int4 int4
int4");
if (mt == NULL) {
    return; /* error */
}

/* create a message and append fields */
msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendInt4(msg, 5)) {
    return; /* error */
}
if (!TipcMsgAppendInt4(msg, 2)) {
    return; /* error */
}
if (!TipcMsgAppendInt4(msg, 9)) {
    return; /* error */
}
```

```

/* create a message file */
msg_file = TipcMsgFileCreate("test.msg",
T_IPC_MSG_FILE_CREATE_WRITE);
if (msg_file == NULL) {
    return; /* error */
}

if (!TipcMsgFileWrite(msg_file, msg)) {
    return; /* error */
}

if (!TipcMsgDestroy(msg)) {
    return; /* error */
}

if (!TipcMsgFileDestroy(msg_file)) {
    return; /* error */
}

```

This is printed to the message file test.msg:

```
xyz_coord _null 5 2 9
```

If the grammar in the call to TipcMtCreate was changed from "int4 int4 int4" to "verbose" in the above code, this is printed to the message file:

```

xyz_coord _null {
    int4 5
    int4 2
    int4 9
}

```

TipcMtDestroy

Name	TipcMtDestroy — destroy a message type
Synopsis	<pre>T_BOOL TipcMtDestroy(mt) T_IPC_MT mt;</pre>
Arguments	<i>mt</i> — message type to destroy
Return Values	TRUE if the message type was successfully destroyed, FALSE otherwise.
Diagnostics	If TipcMtDestroy fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>mt</i> was null
Description	TipcMtDestroy destroys a message type. All memory for the message type is freed. It is unusual to destroy a message type; TipcMtDestroy is provided for completeness.
Caution	None
See Also	TipcMtCreate
Examples	This example looks up a message type and destroys it: <pre>T_IPC_MT mt; mt = TipcMtLookup("numeric_data"); if (mt == NULL) { return; /* error */ } if (!TipcMtDestroy(mt)) { return; /* error */ }</pre>

TipcMtGetCompression

Name	TipcMtGetCompression — get the compression status of a message type
Synopsis	<pre>T_BOOL TipcMtGetCompression(mt, compress_return) T_IPC_MT mt; T_BOOL *compress_return;</pre>
Arguments	<i>mt</i> — message type to get compression status of <i>compress_return</i> — storage for message type compression status
Return Values	TRUE if the compression status was successfully retrieved from the message type, FALSE otherwise.
Diagnostics	If TipcMtGetCompression fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>mt</i> was null or <i>compress_return</i> was null
Description	TipcMtGetCompression gets the compression status of a message type, or whether or not compression is enabled. The compression status of a message type is the default compression status for all messages of this type. When a message is created, its compression status is initialized to the message type compression status. The compression status of an outgoing message can always be set on a per-message basis, but using message type compression status makes it easier to set the default status for all outgoing messages of a specific type. The standard SmartSockets message types by default all have a compression status of FALSE, but these can be changed. User-defined message types can also use whatever compression status is appropriate. When a message type is first created, its compression status is initialized to FALSE.
Caution	If TipcMtGetCompression returns FALSE, it does not store a value in <i>compress_return</i> .
See Also	TipcMtSetCompression

Examples This example gets the compression status of the INFO message type:

```
T_IPC_MT mt;
T_BOOL compress;

mt = TipcMtLookup("info");
if (mt == NULL) {
    return; /* error */
}
if (!TipcMtGetCompression(mt, &compress)) {
    return; /* error */
}
```

TipcMtGetDeliveryMode

Name	TipcMtGetDeliveryMode — get the delivery mode of a message type
Synopsis	<pre>T_BOOL TipcMtGetDeliveryMode(<i>mt</i>, <i>delivery_mode_return</i>) T_IPC_MT <i>mt</i>; T_IPC_DELIVERY_MODE *<i>delivery_mode_return</i>;</pre>
Arguments	<i>mt</i> — message type to get delivery mode from <i>delivery_mode_return</i> — storage for message type delivery mode
Return Values	TRUE if the delivery mode was successfully retrieved from the message type, FALSE otherwise.
Diagnostics	If TipcMtGetDeliveryMode fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>mt</i> was null or <i>delivery_mode_return</i> was null
Description	TipcMtGetDeliveryMode gets the delivery mode of a message type. The delivery mode of a message type is used as the default delivery mode for messages of this type. When a message is created, its delivery mode is initialized to the message type delivery mode. The default message delivery mode is T_IPC_DELIVERY_BEST_EFFORT. In this mode, no special actions are taken to ensure delivery of sent messages. The message is delivered unless network failures or process failures occur. If the message is not delivered, there is no way for the sender to know that delivery failed. When there is a failure, it is possible for some messages to be lost or be delivered in a different order than the order in which they were published. In the case of a network or process failure, a more useful delivery mode is T_IPC_DELIVERY_ORDERED. Messages can still be lost in the event of a failure, but all delivered messages are received in the order in which they were published. Because Java Message Service (JMS) cannot handle messages that come out of order, you must use the T_IPC_DELIVERY_ORDERED delivery mode or one of the GMD delivery modes if you are sending messages to a JMS client. The difference between the other delivery modes and the GMD delivery modes is that in GMD, a copy of the message is kept when the message is sent and acknowledgements are sent when the message is received. No acknowledgements (ACKs) are used in the ordered or best effort delivery modes, and no copy of the message is kept.

One of the GMD delivery modes is `T_IPC_DELIVERY_SOME`. In this mode, the sending process saves a copy of the message in the connection GMD area until the message is successfully delivered, and the sender can also resend the message if necessary. Delivery is considered successful if the sent message is acknowledged by at least one receiving process.

The other and most robust GMD delivery mode is `T_IPC_DELIVERY_ALL`. In this mode, the sending process saves a copy of the message in the connection GMD area until the message is successfully delivered, and the sender can also resend the message if necessary. Delivery is not considered successful until all receiving processes acknowledge the sent message. For two processes communicating through a non-RTclient and non-RTserver `T_IPC_CONN` connection, `T_IPC_DELIVERY_SOME` and `T_IPC_DELIVERY_ALL` are identical because there is only one process receiving the message. For RTclients, the two modes do differ if more than one RTclient is subscribing to the subject in the destination of the message.

The delivery mode of an outgoing message can always be set on a per-message basis, but using message type delivery modes makes it easier to change the default delivery mode for all outgoing messages of a specific type. The standard SmartSockets message types by default usually have a delivery mode of `T_IPC_DELIVERY_BEST_EFFORT`, but these can be changed. The exception is the SmartSockets JMS message types, which have a delivery mode of `T_IPC_DELIVERY_ORDERED`, but that can also be changed. User-defined message types can also use whatever delivery mode is appropriate.

Caution If `TipcMtGetDeliveryMode` returns `FALSE`, it does not store a value in *delivery_mode_return*.

See Also `TipcMsgGetDeliveryMode`, `TipcMtSetDeliveryMode`

Examples This example prints the delivery mode of the NUMERIC_DATA message type:

```
T_IPC_MT mt;
T_IPC_DELIVERY_MODE delivery_mode;
T_STR delivery_mode_str;

mt = TipcMtLookup("numeric_data");
if (mt == NULL) {
    return; /* error */
}

if (!TipcMtGetDeliveryMode(mt, &delivery_mode)) {
    return; /* error */
}

/* convert the delivery mode to a string to make it easier to read */
if (!TipcDeliveryModeToStr(delivery_mode, &delivery_mode_str)) {
    return; /* error */
}

TutOut("NUMERIC_DATA msg type has delivery mode %s\n",
      delivery_mode_str);
```

TipcMtGetDeliveryTimeout

Name	TipcMtGetDeliveryTimeout — get the delivery timeout of a message type
Synopsis	<pre>T_BOOL TipcMtGetDeliveryTimeout(mt, delivery_timeout_return) T_IPC_MT mt; T_REAL8 *delivery_timeout_return;</pre>
Arguments	<i>mt</i>— message type to get delivery timeout from <i>delivery_timeout_return</i> — storage for message type delivery timeout
Return Values	TRUE if the delivery timeout was successfully retrieved from the message type, FALSE otherwise.
Diagnostics	If TipcMtGetDeliveryTimeout fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> was null or <i>delivery_timeout_return</i> was null • T_ERR_DOESNT_EXIST — the timeout has not yet been set for this message type. This is a benign error. The timeout defaults to the connection’s delivery timeout when a message created with this message type is sent.
Description	TipcMtGetDeliveryTimeout gets the delivery timeout of a message type. The delivery timeout of a message controls how long to wait for guaranteed message delivery to complete when a message is sent through a connection with TipcConnMsgSend. The delivery timeout of a message type is used as the default delivery timeout for messages of this type. When a message is created, its delivery timeout is initialized to the message type delivery timeout. See TipcMsgGetDeliveryTimeout for more information on message delivery timeouts. The delivery timeout of an outgoing message can always be set on a per-message basis, but using message type delivery timeouts makes it easier to change the default delivery timeout for all outgoing messages of a specific type. The standard SmartSockets message types by default all have a delivery timeout of unknown, but these can be changed. User-defined message types can also use whatever delivery timeout is appropriate.
Caution	If TipcMtGetDeliveryTimeout returns FALSE, it does not store a value in <i>delivery_timeout_return</i> .
See Also	TipcMtSetDeliveryTimeout, TipcMsgGetDeliveryTimeout

Examples This example gets the delivery timeout of a message type:

```
T_REAL8 delivery_timeout;  
  
if (!TipcMtGetDeliveryTimeout(msg, &delivery_timeout)) {  
    return; /* error */  
}
```

TipcMtGetGrammar

Name	TipcMtGetGrammar — get the grammar of a message type
Synopsis	<pre>T_BOOL TipcMtGetGrammar(mt, grammar_return) T_IPC_MT mt; T_STR *grammar_return;</pre>
Arguments	<i>mt</i> — message type to get grammar from <i>grammar_return</i> — storage for message type grammar
Return Values	TRUE if the grammar was successfully retrieved from the message type, FALSE otherwise.
Diagnostics	If TipcMtGetGrammar fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>mt</i> was null or <i>grammar_return</i> was null
Description	TipcMtGetGrammar gets the grammar of a message type. Message type grammars are lists of field types and are used to write messages to message files in a compact format. See the <i>TIBCO SmartSockets User's Guide</i> for more information on message type grammars.
Caution	If TipcMtGetGrammar returns FALSE, it does not store a value in <i>grammar_return</i>. The value stored in <i>grammar_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcMtCreate, TipcMtGetName, TipcMtGetNum
Examples	This example prints the grammar of the NUMERIC_DATA message type: <pre>T_IPC_MT mt; T_STR grammar; mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA); if (mt == NULL) { return; /* error */ } if (!TipcMtGetGrammar(mt, &grammar)) { return; /* error */ } TutOut("NUMERIC_DATA msg type has grammar %s\n", grammar);</pre>

TipcMtGetHeaderStrEncode

Name	TipcMtGetHeaderStrEncode — get the header string encode property of a message type
Synopsis	<pre>T_BOOL TipcMtGetHeaderStrEncode(mt, header_str_encode_return) T_IPC_MT mt; T_BOOL *header_str_encode_return;</pre>
Arguments	<i>mt</i> — message type to get header string encode property from <i>header_str_encode_return</i> — storage for message type header string encode property
Return Values	TRUE if the header string encode property was successfully retrieved from the message type, FALSE otherwise.
Diagnostics	If TipcMtGetHeaderStrEncode fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>mt</i> was null or <i>header_str_encode_return</i> was null
Description	TipcMtGetHeaderStrEncode gets the header string encode property of a message type. The header string encode property of a message controls whether the message header string properties are converted into four-byte integers when the message is sent through a connection with TipcConnMsgSend. The header string encode property of a message type is used as the default header string encode property for messages of this type. When a message is created, its header string encode property is initialized to the message type header string encode property. See TipcMsgGetHeaderStrEncode for more information on the message header string encode property. The header string encode property of an outgoing message can always be set on a per-message basis, but using the message type header string encode property makes it easier to change the default header string encode property for all outgoing messages of a specific type. The standard SmartSockets message types by default all have a header string encode property of FALSE, but these can be changed. User-defined message types can also use whatever header string encode property is appropriate.
Caution	If TipcMtGetHeaderStrEncode returns FALSE, it does not store a value in <i>header_str_encode_return</i> .
See Also	TipcMtGetHeaderStrEncode, TipcMtSetHeaderStrEncode

Examples This example prints the header string encode property of the NUMERIC_DATA message type:

```
T_IPC_MT mt;
T_BOOL header_str_encode;

mt = TipcMtLookup("numeric_data");
if (mt == NULL) {
    return; /* error */
}

if (!TipcMtGetHeaderStrEncode(mt, &header_str_encode)) {
    return; /* error */
}

TutOut("NUMERIC_DATA msg type has header string encode property
%s\n",
      header_str_encode ? "TRUE" : "FALSE");
```

TipcMtGetLbMode

Name	TipcMtGetLbMode — get the load balancing mode of a message type
Synopsis	<pre>T_BOOL TipcMtGetLbMode(mt, lb_mode_return) T_IPC_MT mt; T_IPC_LB_MODE *lb_mode_return;</pre>
Arguments	<i>mt</i>— message type to get load balancing mode from <i>lb_mode_return</i> — storage for message type load balancing mode
Return Values	TRUE if the load balancing mode was successfully retrieved from the message type, FALSE otherwise.
Diagnostics	If TipcMtGetLbMode fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>mt</i> was null or <i>lb_mode_return</i> was null
Description	TipcMtGetLbMode gets the load balancing mode of a message type. The load balancing mode of a message controls how (or if) the message is load balanced among RTclients subscribed to a load balanced subject. The load balancing mode of a message type is used as the default load balancing mode for messages of this type. When a message is created, its load balancing mode is initialized to the message type load balancing mode. See TipcMsgGetLbMode for more information on message load balancing modes. The load balancing mode of an outgoing message can always be set on a per-message basis, but using message type load balancing modes makes it easier to change the default load balancing mode for all outgoing messages of a specific type. The standard SmartSockets message types by default all have a load balancing mode of T_IPC_LB_NONE, but these can be changed. User-defined message types can also use whatever load balancing mode is appropriate.
Caution	If TipcMtGetLbMode returns FALSE, it does not store a value in <i>lb_mode_return</i> .
See Also	TipcMtSetLbMode, TipcMtGetLbMode

Examples This example prints the load balancing mode of the NUMERIC_DATA message type:

```
T_IPC_MT mt;
T_IPC_LB_MODE lb_mode;
T_STR lb_mode_str;

mt = TipcMtLookup("numeric_data");
if (mt == NULL) {
    return; /* error */
}

if (!TipcMtGetLbMode(mt, &lb_mode)) {
    return; /* error */
}

/* convert the load balancing mode to a string to make it easier to read */
if (!TipcLbModeToStr(lb_mode, &lb_mode_str)) {
    return; /* error */
}

TutOut("NUMERIC_DATA msg type has load balancing mode %s\n",
        lb_mode_str);
```

TipcMtGetName

Name	TipcMtGetName — get the name of a message type
Synopsis	<pre>T_BOOL TipcMtGetName(mt, name_return) T_IPC_MT mt; T_STR *name_return;</pre>
Arguments	<i>mt</i> — message type to get name from <i>name_return</i> — storage for message type name
Return Values	TRUE if the name was successfully retrieved from the message type, FALSE otherwise.
Diagnostics	If TipcMtGetName fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>mt</i> was null or <i>name_return</i> was null
Description	TipcMtGetName gets the name of a message type. Message type names are identifiers.
Caution	If TipcMtGetName returns FALSE, it does not store a value in <i>name_return</i>. The value stored in <i>name_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcMtCreate, TipcMtGetGrammar, TipcMtGetNum
Examples	This example prints the name of the NUMERIC_DATA message type: <pre>T_IPC_MT mt; T_STR name; mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA); if (mt == NULL) { return; /* error */ } if (!TipcMtGetName(mt, &name)) { return; /* error */ } TutOut("NUMERIC_DATA msg type has name %s\n", name);</pre>

TipcMtGetNum

Name	TipcMtGetNum — get the number of a message type
Synopsis	<pre>T_BOOL TipcMtGetNum(mt, num_return) T_IPC_MT mt; T_INT4 *num_return;</pre>
Arguments	<i>mt</i> — message type to get number from <i>num_return</i> — storage for message type number
Return Values	TRUE if the number was successfully retrieved from the message type, FALSE otherwise.
Diagnostics	If TipcMtGetNum fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>mt</i> was null or <i>num_return</i> was null
Description	TipcMtGetNum gets the number of a message type. Message type numbers less than 1 are reserved for SmartSockets standard message types.
Caution	If TipcMtGetNum returns FALSE, it does not store a value in <i>num_return</i> .
See Also	TipcMtCreate, TipcMtGetGrammar, TipcMtGetName
Examples	This example prints the number of the NUMERIC_DATA message type: <pre>T_IPC_MT mt; T_INT4 num; mt = TipcMtLookup("numeric_data"); if (mt == NULL) { return; /* error */ } if (!TipcMtGetNum(mt, &num)) { return; /* error */ } TutOut("NUMERIC_DATA msg type has num %d\n", num);</pre>

TipcMtGetPriority

Name	TipcMtGetPriority — get the priority of a message type
Synopsis	<pre>T_BOOL TipcMtGetPriority(mt, priority_return) T_IPC_MT mt; T_INT2 *priority_return;</pre>
Arguments	<i>mt</i> — message type to get priority from <i>priority_return</i> — storage for message type priority
Return Values	TRUE if the priority was successfully retrieved from the message type, FALSE otherwise.
Diagnostics	If TipcMtGetPriority fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> was null or <i>priority_return</i> was null • T_ERR_VAL_UNKNOWN — the priority of the message type is unknown (not set)
Description	TipcMtGetPriority gets the priority of a message type. The priority of a message type, if set, is used as the default priority for messages of this type. The message type priority can either be set to a specific value or it can be unknown. When a message is created, its priority is initialized to the message type priority (if set) or to the value of the option Default_Msg_Priority (if the message type priority is unknown). See TipcMsgSetPriority for more information on message priorities. The priority of an outgoing message can always be set on a per-message basis, but using message type priorities makes it easier to raise or lower the default priority for all outgoing messages of a specific type. Note that if set, the message type priority always overrides the value in the option Default_Msg_Priority. The standard SmartSockets message types by default all have a priority of unknown, but these can be changed. User-defined message types can also use whatever priority is appropriate. When a message type is first created, its priority is initialized to unknown.
Caution	If TipcMtGetPriority returns FALSE, it does not store a value in <i>priority_return</i> .
See Also	TipcMtSetPriority, TipcMtSetPriorityUnknown

Examples This example prints the priority of the NUMERIC_DATA message type:

```
T_IPC_MT mt;
T_INT2 priority;

mt = TipcMtLookup("numeric_data");
if (mt == NULL) {
    return; /* error */
}

if (!TipcMtGetPriority(mt, &priority)) {
    if (TutErrNumGet() == T_ERR_VAL_UNKNOWN) {
        TutOut("NUMERIC_DATA msg type does not have priority set.\n");
    }
    else {
        return; /* error */
    }
}
else {
    TutOut("NUMERIC_DATA msg type has priority %d\n", priority);
}
```


TipcMtGetUserProp

Name	TipcMtGetUserProp — get the user-defined property of a message type
Synopsis	<pre>T_BOOL TipcMtGetUserProp(mt, user_prop_return) T_IPC_MT mt; T_INT4 *user_prop_return;</pre>
Arguments	<i>mt</i> — message type to get user-defined property from <i>user_prop_return</i> — storage for message type user-defined property
Return Values	TRUE if the user-defined property was successfully retrieved from the message type, FALSE otherwise.
Diagnostics	If TipcMtGetUserProp fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>mt</i> was null or <i>user_prop_return</i> was null
Description	TipcMtGetUserProp gets the user-defined property of a message type. The user-defined property of a message type is used as the default user-defined property for messages of this type. When a message is created, its user-defined property is initialized to the message type user-defined property. You can use the user-defined property for any purpose, such as attaching a version number to message types. This property is not used internally by SmartSockets. When a message is sent through a connection with TipcConnMsgSend, the user-defined property is byte-swapped along with all the other integer message properties. On architectures where a pointer is four bytes, a pointer can be stored in the user-defined property of a message type. This is useful for attaching extra temporary data to the message type. Use pointers with caution; when a message is sent through a connection, the data the pointer refers to does not exist in the receiving process. In this situation, add extra fields to the message instead of using a pointer in the user-defined property. The user-defined property of an outgoing message can always be set on a per-message basis, but using message type user-defined properties makes it easier to change the default user-defined property for all outgoing messages of a specific type. The standard SmartSockets message types by default all have a user-defined property of 0, but these can be changed. User-defined message types can also use whatever user-defined property is appropriate.
Caution	If TipcMtGetUserProp returns FALSE, it does not store a value in <i>user_prop_return</i> .

See Also TipcMsgGetUserProp, TipcMtSetUserProp

Examples This example prints the user-defined property of the NUMERIC_DATA message type:

```
T_IPC_MT mt;
T_INT4 user_prop;

mt = TipcMtLookup("numeric_data");
if (mt == NULL) {
    return; /* error */
}

if (!TipcMtGetUserProp(mt, &user_prop)) {
    return; /* error */
}

TutOut("NUMERIC_DATA msg type has user prop %d\n", user_prop);
```

TipcMtLogAddMt

Name	TipcMtLogAddMt — add a message type to a message file logging type
Synopsis	<pre>T_BOOL TipcMtLogAddMt(log_type, mt) T_IPC_MT_LOG_TYPE log_type; T_IPC_MT mt;</pre>
Arguments	<i>log_type</i> — message file logging type <i>mt</i> — message type to add to list
Return Values	TRUE if the message type was successfully added, FALSE otherwise.
Diagnostics	If TipcMtLogAddMt fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR -- <i>mt</i> was null • T_ERR_VAL_INVALID -- <i>log_type</i> was not a valid T_IPC_MT_LOG_TYPE • T_ERR_ALREADY_EXISTS -- <i>mt</i> already exists in the list • any error number from TutPtrAryAppend
Description	TipcMtLogAddMt adds a message type to a message file logging type. The logging type T_IPC_MT_LOG_DATA is controlled by the options Log_In_Data and Log_Out_Data. The logging type T_IPC_MT_LOG_STATUS is controlled by the options Log_In_Status and Log_Out_Status. The logging type T_IPC_MT_LOG_INTERNAL is controlled by the options Log_In_Internal and Log_Out_Internal. All of the standard SmartSockets message types are added automatically to one logging type. Message types can be added to multiple logging types.
Caution	TipcMtLogAddMt only takes effect if called before the RTclient has created a connection to RTserver.
See Also	TipcMtLogRemoveMt; see the <i>TIBCO SmartSockets Utilities</i> for information on TutPtrAryAppend.

Examples This example creates a message type and adds it to the data logging type with TipcMtLogAddMt:

```
#define USER_MT_COORD_DATA 100

mt = TipcMtCreate("coord_data", USER_MT_COORD_DATA,
                 "int4 /*x*/ int4 /*y*/ int4 /*z*/");
if (mt == NULL) {
    return; /* error */
}

if (!TipcMtLogAddMt(T_IPC_MT_LOG_DATA, mt)) {
    return; /* error */
}

TutCommandParseStr("setopt log_in_data data.msg");
/*
Data messages that arrive from RTserver (include those of type
USER_MT_COORD_DATA) will now be logged to the file data.msg.
*/
```

TipcMtLogRemoveMt

Name	TipcMtLogRemoveMt — remove a message type from a message file logging type
Synopsis	<pre>T_BOOL TipcMtLogRemoveMt(log_type, mt) T_IPC_MT_LOG_TYPE log_type; T_IPC_MT mt;</pre>
Arguments	<i>log_type</i> — message file logging type <i>mt</i> — message type to remove from list
Return Values	TRUE if message type was successfully removed, FALSE otherwise.
Diagnostics	If TipcMtLogRemoveMt fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR -- <i>mt</i> was null • T_ERR_VAL_INVALID -- <i>log_type</i> was not a valid T_IPC_SRV_LOG_TYPE • any error number from TutPtrAryRemove
Description	TipcMtLogRemoveMt removes a message type from a message file logging type. See TipcMtLogAddMt for more information on logging types.
Caution	TipcMtLogRemoveMt only takes effect if called before the RTclient has created a connection to RTserver.
See Also	TipcMtLogAddMt; see the <i>TIBCO SmartSockets Utilities</i> for information on TutPtrAryRemove.
Examples	This example removes the TIME message type from the data logging type with TipcMtLogRemoveMt: <pre>mt = TipcMtLookupByNum(T_MT_TIME); if (mt == NULL) { return; /* error */ } if (!TipcMtLogRemoveMt(T_IPC_MT_LOG_DATA, mt)) { return; /* error */ }</pre>

TipcMtLookup

Name	TipcMtLookup — look up a message type by name
Synopsis	<pre>T_IPC_MT TipcMtLookup(name) T_STR name;</pre>
Arguments	<i>name</i> — name of message type (not case-sensitive)
Return Values	Message type if successful, NULL otherwise.
Diagnostics	If TipcMtLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>name</i> was null • any error number from TutHashLookup
Description	TipcMtLookup looks up a message type with the desired name. The standard SmartSockets message types use C #define defines for numbers that are similar to the names. (For example, the message type with name "numeric_data" has a defined number T_MT_NUMERIC_DATA.)
Caution	None
See Also	TipcMtCreate, TipcMtDestroy, TipcMtLookupByNum; see <i>TIBCO SmartSockets Utilities</i> for information on TutHashLookup.
Examples	This example looks up the ALERT message type by name: <pre>T_IPC_MT mt; mt = TipcMtLookup("alert"); if (mt == NULL) { return; /* error */ }</pre>

TipcMtLookupByNum

Name	TipcMtLookupByNum — look up a message type by number
Synopsis	<pre>T_IPC_MT TipcMtLookupByNum(<i>num</i>) T_INT4 <i>num</i>;</pre>
Arguments	<i>num</i> — number of message type
Return Values	Message type if successful, NULL otherwise.
Diagnostics	If TipcMtLookupByNum fails, it returns NULL and sets the global SmartSockets error number to: any error number from TutHashLookup
Description	TipcMtLookupByNum looks up a message type with the desired number. The standard SmartSockets message types use C #define defines for numbers that are similar to the names. (For example, the message type with name "numeric_data" has a defined number T_MT_NUMERIC_DATA.)
Caution	None
See Also	TipcMtCreate, TipcMtDestroy, TipcMtLookup; see <i>TIBCO SmartSockets Utilities</i> for information on TutHashLookup.
Examples	This example looks up the ALERT message type by number: <pre>T_IPC_MT mt; mt = TipcMtLookupByNum(T_MT_ALERT); if (mt == NULL) { return; /* error */ }</pre>

TipcMtPrint

Name	TipcMtPrint — print all information about a message type in portable, reproducible format
Synopsis	<pre>T_BOOL TipcMtPrint(<i>mt</i>, <i>func</i>) T_IPC_MT <i>mt</i>; T_OUT_FUNC <i>func</i>;</pre>
Arguments	<i>mt</i> — message type to print <i>func</i> — TutOut-style output function
Return Values	TRUE if the message type was successfully printed, FALSE otherwise.
Diagnostics	If TipcMtPrint fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>mt</i> was null or <i>func</i> was null
Description	TipcMtPrint prints all information about a message type in a portable, reproducible format to a TutOut-style output function. The output function is called once for each line of output. TipcMtPrint first prints the name, number, and grammar string of the message type. TipcMtPrint then prints an internal representation of the grammar.
Caution	None
See Also	TipcMtPrint

Examples This example prints a few message types using TutOut:

```
T_IPC_MT mt;

mt = TipcMtLookup("numeric_data");
if (mt == NULL) {
    return; /* error */
}
if (!TipcMtPrint(mt, TutOut)) {
    return; /* error */
}

mt = TipcMtLookup("subject_set_subscribe");
if (mt == NULL) {
    return; /* error */
}
if (!TipcMtPrint(mt, TutOut)) {
    return; /* error */
}
```

The above fragment prints this output:

```
name = numeric_data
num = -22
grammar string = { id real8 }
priority = unknown
delivery_mode = best_effort
user_prop = 0
grammar:
    type group
        type id
        type real8
name = subject_set_subscribe
num = -76
grammar string = id /*subject*/ bool /*subscribe_flag*/ bool
/*lb_flag*/
priority = unknown
delivery_mode = best_effort
user_prop = 0
grammar:
    type id
    type bool
    type bool
```

TipcMtSetCompression

Name	TipcMtSetCompression — set the compression status of a message type
Synopsis	<pre>T_BOOL TipcMtSetCompression(mt, compress) T_IPC_MT mt; T_BOOL compress;</pre>
Arguments	<i>mt</i> — message type to set compression status for <i>compress</i> — the new message type compression status (TRUE to enable compression, FALSE to disable)
Return Values	TRUE if the compression status was successfully set for the message type, FALSE otherwise.
Diagnostics	If TipcMtSetCompression fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> was null • T_ERR_VAL_INVALID — <i>compress</i> was not TRUE or FALSE
Description	TipcMtSetCompression sets the compression status of a message type. The compression status of a message type is the default compression status for all messages of this type. When a message is created, its compression status is initialized to the message type compression status. The compression status of an outgoing message can always be set on a per-message basis, but using message type compression status makes it easier to set the default status for all outgoing messages of a specific type. Use TipcMsgSetCompression to override the message type compression setting on a per-message basis. The standard SmartSockets message types by default all have a compression status of FALSE, but these can be changed. User-defined message types can also use whatever compression status is appropriate. TRUE enables compression for the message type; FALSE disables compression. The default compression setting for all message types is FALSE.

Caution Do not send compressed messages to an RT process running on a SmartSockets software release lower than 6.5.0.

Sending a compressed message to an RT process running an earlier version of SmartSockets may result in unexpected behavior. Typically, the RT process running on an earlier version of SmartSockets returns an error code of `T_ERR_FT_MISMATCH` or `T_ERR_FT_INVALID` when attempting to access the fields of a compressed message. However, it is also possible for the RT process to retrieve invalid data from the field without generating an error.

See Also `TipcMsgSetCompression`, `TipcMtGetCompression`

Examples This example sets the compression status of the INFO message type TRUE:

```
T_IPC_MT mt;

mt = TipcMtLookup("info");
if (mt == NULL) {
    return; /* error */
}

if (!TipcMtSetCompression(mt, T_TRUE)) {
    return; /* error */
}
```

TipcMtSetDeliveryMode

Name	TipcMtSetDeliveryMode — set the delivery mode of a message type
Synopsis	<pre>T_BOOL TipcMtSetDeliveryMode(mt, delivery_mode) T_IPC_MT mt; T_IPC_DELIVERY_MODE delivery_mode;</pre>
Arguments	<i>mt</i> — message type to set delivery mode for <i>delivery_mode</i> — message type delivery mode (T_IPC_DELIVERY_BEST_EFFORT, T_IPC_DELIVERY_ORDERED, T_IPC_DELIVERY_SOME, or T_IPC_DELIVERY_ALL)
Return Values	TRUE if the delivery mode was successfully set for the message type, FALSE otherwise.
Diagnostics	If TipcMtSetDeliveryMode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> was null • T_ERR_VAL_INVALID — <i>delivery_mode</i> was not a valid T_IPC_DELIVERY_MODE
Description	TipcMtSetDeliveryMode sets the delivery mode of a message type. The delivery mode of a message type is used as the default delivery mode for messages of this type. See TipcMtGetDeliveryMode for more information on message type delivery modes.
Caution	None
See Also	TipcMsgSetDeliveryMode, TipcMtGetDeliveryMode
Examples	This example sets the delivery mode of the NUMERIC_DATA message type to T_IPC_DELIVERY_ALL: <pre>T_IPC_MT mt; mt = TipcMtLookup("numeric_data"); if (mt == NULL) { return; /* error */ } if (!TipcMtSetDeliveryMode(mt, T_IPC_DELIVERY_ALL)) { return; /* error */ }</pre> <i>/* After this point, all newly-created NUMERIC_DATA messages will have a default delivery mode of T_IPC_DELIVERY_ALL. */</i>

TipcMtSetDeliveryTimeout

Name	TipcMtSetDeliveryTimeout — set the delivery timeout of a message type
Synopsis	<pre>T_BOOL TipcMtSetDeliveryTimeout(<i>mt</i>, <i>delivery_timeout</i>) T_IPC_MT <i>mt</i>; T_REAL8 <i>delivery_timeout</i>;</pre>
Arguments	<i>mt</i> — message type to set delivery timeout for <i>delivery_timeout</i> — message type delivery timeout in seconds
Return Values	TRUE if the delivery timeout was successfully set for the message type, FALSE otherwise.
Diagnostics	If TipcMtSetDeliveryTimeout fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> was null • T_ERR_VAL_TOO_SMALL — <i>delivery_timeout</i> was negative
Description	TipcMtSetDeliveryTimeout sets the delivery timeout of a message type. The delivery timeout of a message type is used as the default delivery timeout for messages of this type. See TipcMtGetDeliveryTimeout for more information on message type delivery timeouts.
Caution	None
See Also	TipcMsgSetDeliveryTimeout, TipcMtGetDeliveryTimeout
Examples	This example sets the delivery timeout of the NUMERIC_DATA message type to 5 seconds: <pre>T_IPC_MT <i>mt</i>; <i>mt</i> = TipcMtLookup("numeric_data"); if (<i>mt</i> == NULL) { return; /* error */ } if (!TipcMtSetDeliveryTimeout(<i>mt</i>, 5.0)) { return; /* error */ } /* After this point, all newly-created NUMERIC_DATA messages will have a default delivery timeout of 5 seconds. */</pre>

TipcMtSetHeaderStrEncode

Name	TipcMtSetHeaderStrEncode — set the header string encode property of a message type
Synopsis	<pre>T_BOOL TipcMtSetHeaderStrEncode(mt, header_str_encode) T_IPC_MT mt; T_BOOL header_str_encode;</pre>
Arguments	<i>mt</i> — message type to set header string encode property for <i>header_str_encode</i> — message type header string encode property
Return Values	TRUE if the header string encode property was successfully set for the message type, FALSE otherwise.
Diagnostics	If TipcMtSetHeaderStrEncode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> was null • T_ERR_VAL_INVALID — <i>header_str_encode</i> was not TRUE or FALSE
Description	TipcMtSetHeaderStrEncode sets the header string encode property of a message type. The header string encode property of a message type is used as the default header string encode property for messages of this type. See TipcMtGetHeaderStrEncode for more information on the message type header string encode property.
Caution	None
See Also	TipcMsgSetHeaderStrEncode, TipcMtGetHeaderStrEncode
Examples	This example enables the header string encode property of the NUMERIC_DATA message type: <pre>T_IPC_MT mt; mt = TipcMtLookup("numeric_data"); if (mt == NULL) { return; /* error */ } if (!TipcMtSetHeaderStrEncode(mt, TRUE)) { return; /* error */ } /* After this point, all newly-created NUMERIC_DATA messages will have a default header string encode property of TRUE. */</pre>

TipcMtSetLbMode

Name	TipcMtSetLbMode — set the load balancing mode of a message type
Synopsis	<pre>T_BOOL TipcMtSetLbMode(mt, lb_mode) T_IPC_MT mt; T_IPC_LB_MODE lb_mode;</pre>
Arguments	<i>mt</i> — message type to set load balancing mode for <i>lb_mode</i> — message type load balancing mode (T_IPC_LB_NONE, T_IPC_LB_ROUND_ROBIN, T_IPC_LB_WEIGHTED, or T_IPC_LB_SORTED)
Return Values	TRUE if the load balancing mode was successfully set for the message type, FALSE otherwise.
Diagnostics	If TipcMtSetLbMode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> was null • T_ERR_VAL_INVALID — <i>lb_mode</i> was not a valid T_IPC_LB_MODE
Description	TipcMtSetLbMode sets the load balancing mode of a message type. The load balancing mode of a message type is used as the default load balancing mode for messages of this type. See TipcMtGetLbMode for more information on message type load balancing modes.
Caution	None
See Also	TipcMsgSetLbMode, TipcMtGetLbMode
Examples	This example sets the load balancing mode of the NUMERIC_DATA message type to T_IPC_LB_WEIGHTED: <pre>T_IPC_MT mt; mt = TipcMtLookup("numeric_data"); if (mt == NULL) { return; /* error */ } if (!TipcMtSetLbMode(mt, T_IPC_LB_WEIGHTED)) { return; /* error */ } /* After this point, all newly-created NUMERIC_DATA messages will have a default load balancing mode of T_IPC_LB_WEIGHTED. */</pre>

TipcMtSetPriority

Name	TipcMtSetPriority — set the priority of a message type
Synopsis	<pre>T_BOOL TipcMtSetPriority(<i>mt</i>, <i>priority</i>) T_IPC_MT <i>mt</i>; T_INT2 <i>priority</i>;</pre>
Arguments	<i>mt</i> — message type to set priority for <i>priority</i> — message type priority
Return Values	TRUE if the priority was successfully set for the message type, FALSE otherwise.
Diagnostics	If TipcMtSetPriority fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>mt</i> was null
Description	TipcMtSetPriority sets the priority of a message type. The priority of a message type, if set, is used as the default priority for messages of this type. See TipcMtGetPriority for more information on message type priorities. Use the function TipcMtSetPriorityUnknown to set the message type priority to unknown.
Caution	Setting message priorities on GMD or load-balanced (LB) messages is not recommended. GMD and LB rely on high sequence numbers for correct functioning, and unexpected behavior may occur if such messages are received out of order due to the use of message priorities.
See Also	TipcMsgSetPriority, TipcMtGetPriority, TipcMtSetPriorityUnknown
Examples	This example sets the priority of the NUMERIC_DATA message type to 2: <pre>T_IPC_MT <i>mt</i>; <i>mt</i> = TipcMtLookup("numeric_data"); if (<i>mt</i> == NULL) { return; /* error */ } if (!TipcMtSetPriority(<i>mt</i>, 2)) { return; /* error */ }</pre>

TipcMtSetPriorityUnknown

Name	TipcMtSetPriorityUnknown — set the priority of a message type to unknown
Synopsis	<pre>T_BOOL TipcMtSetPriorityUnknown(mt) T_IPC_MT mt;</pre>
Arguments	<i>mt</i> — message type to set priority to unknown
Return Values	TRUE if the priority was successfully set for the message type, FALSE otherwise.
Diagnostics	If TipcMtSetPriorityUnknown fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>mt</i> was null
Description	TipcMtSetPriorityUnknown sets the priority of a message type to unknown. This can also be thought of as unsetting the priority. The priority of a message type, if set, is used as the default priority for messages of this type. See TipcMtGetPriority for more information on message type priorities. Use the function TipcMtSetPriority to set the message type priority to a specific value.
Caution	None
See Also	TipcMtGetPriority, TipcMtSetPriority
Examples	This example unsets the priority of the NUMERIC_DATA message type: <pre>T_IPC_MT mt; mt = TipcMtLookup("numeric_data"); if (mt == NULL) { return; /* error */ } if (!TipcMtSetPriorityUnknown(mt)) { return; /* error */ }</pre>

TipcMtSetUserProp

Name	TipcMtSetUserProp — set the user-defined property of a message type
Synopsis	<pre>T_BOOL TipcMtSetUserProp(mt, user_prop) T_IPC_MT mt; T_INT4 user_prop;</pre>
Arguments	<i>mt</i> — message type to set user-defined property for <i>user_prop</i> — message type user-defined property
Return Values	TRUE if the user-defined property was successfully set for the message type, FALSE otherwise.
Diagnostics	If TipcMtSetUserProp fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>mt</i> was null
Description	TipcMtSetUserProp sets the user-defined property of a message type. The user-defined property of a message type is used as the default user-defined property for messages of this type. See TipcMtGetUserProp for more information on message type user-defined properties.
Caution	None
See Also	TipcMsgSetUserProp, TipcMtGetUserProp
Examples	This example sets the user-defined property of the NUMERIC_DATA message type to 42: <pre>T_IPC_MT mt; mt = TipcMtLookup("numeric_data"); if (mt == NULL) { return; /* error */ } if (!TipcMtSetUserProp(mt, 42)) { return; /* error */ }</pre> <i>/* After this point, all newly-created NUMERIC_DATA messages will have a default user-defined property of 42. */</i>

TipcMtTraverse

Name	TipcMtTraverse — traverse all message types
Synopsis	<pre>T_PTR TipcMtTraverse(func, arg) T_IPC_MT_TRAV_FUNC func; T_PTR arg;</pre>
Arguments	<i>func</i> — function to call once for each message type <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	The first non-null return value from <i>func</i> , or NULL if <i>func</i> always returns NULL.
Diagnostics	If TipcMtTraverse fails or traverses all message types, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • any error number from TutHashTraverse
Description	TipcMtTraverse traverses all message types and calls <i>func</i> once for each message type. When <i>func</i> returns a non-null value, then TipcMtTraverse returns with that value.
Caution	The traversal function must always return a value and must return NULL for the traversal to continue. The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes. The order that the message types are traversed is not controllable by users. If TipcMtTraverse returns NULL, it may be because there was an error or simply because <i>func</i> always returned NULL. If <i>func</i> always returns NULL, then the global SmartSockets error number is set to T_ERR_END_OF_TRAVERSAL.
See Also	TipcMtPrint; see <i>TIBCO SmartSockets Utilities</i> for information on TutHashTraverse.

Examples This example prints the names and numbers of all message types:

```
T_PTR T_ENTRY print_one_mt(name, mt, arg)
T_STR name;
T_IPC_MT mt;
T_PTR arg; /* not used */
{
    T_INT4 num;

    /* could use TipcMtGetName, but easier to use name argument */
    TutOut("Message type name is %s.\n", name);

    if (!TipcMtGetNum(mt, &num)) {
        return; /* error */
    }
    TutOut("Message type number is %d.\n", num);

    return NULL; /* continue traversal */
} /* print_one_mt */

/* ===== */
/*...code from calling function is below */

(void)TipcMtTraverse(print_one_mt, NULL);
```

Chapter 12 **TipcProperties***

This chapter covers the TipcProperties API functions:

- *TipcPropertiesCreate*
- *TipcPropertiesCreateMsg*
- *TipcPropertiesDestroy*
- *TipcPropertiesGet*
- *TipcPropertiesGetCount*
- *TipcPropertiesGetDefault*
- *TipcPropertiesMsgCreate*
- *TipcPropertiesSet*
- *TipcPropertiesTraverse*

TipcPropertiesCreate

Name	TipcPropertiesCreate — create a properties table
Synopsis	<code>T_IPC_PROPERTIES TipcPropertiesCreate(void)</code>
Arguments	None
Return Values	Pointer to a properties table if successful, NULL otherwise.
Diagnostics	If TipcPropertiesCreate fails, it returns NULL.
Description	TipcPropertiesCreate returns a pointer to a properties table. Properties tables are used for querying and updating the properties of queues and queue managers.
Caution	None
See Also	TipcPropertiesDestroy, TipcPropertiesCreateMsg, TipcPropertiesMsgCreate
Examples	This example creates a properties table: <pre>T_IPC_PROPERTIES props; /* create a properties table */ props = TipcPropertiesCreate(); if (props == NULL) { return; /* error */ }</pre>

TipcPropertiesCreateMsg

Name	TipcPropertiesCreateMsg — create a properties table from a SmartSockets message
Synopsis	<pre>T_IPC_PROPERTIES TipcPropertiesCreateMsg(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — a SmartSockets message
Return Values	Pointer to a properties table if successful, NULL otherwise.
Diagnostics	If TipcPropertiesCreateMsg fails, it returns NULL and sets the global SmartSockets error code to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message
Description	TipcPropertiesCreateMsg creates a properties table from a SmartSockets message. <i>msg</i> must be a message with name-value pairs. The name-value pairs should be strings appended one after the other. Use the TipcMsgAppendStr function to append strings to a SmartSockets message.
Caution	None
See Also	TipcPropertiesDestroy, TipcPropertiesCreate, TipcPropertiesMsgCreate
Examples	This example creates a properties table from a SmartSockets message: <pre>T_IPC_PROPERTIES props; T_IPC_MSG props_msg; /* create a properties table from a message */ props = TipcPropertiesCreateMsg(props_msg); if (props == NULL) { return; /* error */ }</pre>

TipcPropertiesDestroy

Name	TipcPropertiesDestroy — destroy a properties table
Synopsis	<pre>T_BOOL TipcPropertiesDestroy(props) T_IPC_PROPERTIES props;</pre>
Arguments	<i>props</i> — properties table
Return Values	TRUE if table is destroyed successfully, FALSE otherwise.
Diagnostics	If TipcPropertiesDestroy fails, it returns FALSE and sets the global SmartSockets error code to: T_ERR_NULL_PTR — <i>props</i> was null
Description	TipcPropertiesDestroy frees the memory allocated to the properties table.
Caution	None
See Also	TipcPropertiesCreate, TipcPropertiesCreateMsg, TipcPropertiesMsgCreate
Examples	This example destroys a properties table: <pre>T_IPC_PROPERTIES props; T_BOOL status; /* destroy properties structure */ status = TipcPropertiesDestroy(props); if (!status) { return; /* error */ }</pre>

TipcPropertiesGet

Name	TipcPropertiesGet — get the property value for the specified property name
Synopsis	<pre>T_STR TipcPropertiesGet(props, name) T_IPC_PROPERTIES props; T_STR name;</pre>
Arguments	<i>props</i> — properties table <i>name</i> — property name
Return Values	Properties value if successful, NULL otherwise.
Diagnostics	If TipcPropertiesGet fails, it returns NULL and sets the global SmartSockets error code to one of: • T_ERR_NULL_PTR — <i>props</i> was null or <i>name</i> was null • T_ERR_DOESNT_EXIST — entry for <i>name</i> does not exist in the properties table
Description	TipcPropertiesGet returns the value associated with the specified property name. If <i>name</i> is not found in the table, then TipcPropertiesGet returns NULL.
Caution	None
See Also	TipcPropertiesDestroy, TipcPropertiesCreate, TipcPropertiesSet
Examples	This example gets values from the properties table: <pre>T_IPC_PROPERTIES props; T_BOOL status; status = TipcPropertiesSet(props, "prop1", "val1"); if (!status) { return; /* error */ } TutOut("The Key prop1 : has Value : %s\n", TipcPropertiesGet(props, "prop1"));</pre>

TipcPropertiesGetCount

Name	TipcPropertiesGetCount — get the count of name-value pairs in the properties table
Synopsis	<pre>T_BOOL TipcPropertiesGetCount(<i>props</i>, <i>count_return</i>) T_IPC_PROPERTIES <i>props</i>; T_INT4 *<i>count_return</i>;</pre>
Arguments	<i>props</i> — properties table <i>count_return</i> — pointer to an integer
Return Values	TRUE if count returns successfully, FALSE otherwise.
Diagnostics	If TipcPropertiesGetCount fails, it returns FALSE and sets the global SmartSockets error code to: T_ERR_NULL_PTR — <i>props</i> was null or <i>count_return</i> was null
Description	TipcPropertiesGetCount returns the number of name-value pairs in the properties table.
Caution	None
See Also	TipcPropertiesSet, TipcPropertiesGet, TipcPropertiesGetDefault
Examples	This example gets the number of name-value pairs in the table: <pre>T_IPC_PROPERTIES props; T_BOOL status; T_INT4 count; status = TipcPropertiesGetCount(props, &count); if (!status) { return; /* error */ } TutOut("Num of props = <%d>\n", count);</pre>

TipcPropertiesGetDefault

Name	TipcPropertiesGetDefault — get the default property value for the specified property name
Synopsis	<pre>T_STR TipcPropertiesGetDefault(<i>props</i>, <i>name</i>) T_IPC_PROPERTIES <i>props</i>; T_STR <i>name</i>;</pre>
Arguments	<i>props</i> — properties table <i>name</i> — property name
Return Values	TipcPropertiesGetDefault returns the default value associated with the property name on success and NULL otherwise.
Diagnostics	If TipcPropertiesGetDefault fails, it returns NULL and sets the global SmartSockets error code to one of: • T_ERR_NULL_PTR — <i>props</i> was null or <i>name</i> was null • T_ERR_DOESNT_EXIST — an entry for <i>name</i> does not exist in the properties table
Description	TipcPropertiesGetDefault returns the default value associated with the property name. If the property name is not found then it returns NULL.
Caution	None
See Also	TipcPropertiesGet, TipcPropertiesGetCount, TipcPropertiesSet
Examples	This example gets the default value for a property: <pre>T_IPC_PROPERTIES props; T_STR value_return; value_return = TipcPropertiesGetDefault(props, "prop1"); TutOut("The Key is prop1: Value is %s\n", value_return);</pre>

TipcPropertiesMsgCreate

Name	TipcPropertiesMsgCreate — create a SmartSockets message from a properties table
Synopsis	<pre>T_IPC_MSG TipcPropertiesMsgCreate(props) T_IPC_PROPERTIES props;</pre>
Arguments	<i>props</i> — property table
Return Values	A pointer to a SmartSockets message if successful, NULL otherwise.
Diagnostics	If TipcPropertiesMsgCreate fails, it returns NULL and sets the global SmartSockets error code to: • T_ERR_NULL_PTR — <i>props</i> was null
Description	TipcPropertiesMsgCreate creates a SmartSockets message from a properties table.
Caution	None
See Also	TipcPropertiesCreateMsg
Examples	This example creates a SmartSockets message from a properties table: <pre>T_IPC_PROPERTIES props; T_IPC_MSG props_msg; props_msg = TipcPropertiesMsgCreate(props); if (props_msg == NULL) { return; /* error */ } TipcMsgPrint(props_msg, TutOut);</pre>

TipcPropertiesSet

Name	TipcPropertiesSet — set the specified name and value in the properties table
Synopsis	<pre>T_BOOL TipcPropertiesSet(props, name, value) T_IPC_PROPERTIES props; T_STR name; T_STR value;</pre>
Arguments	<i>props</i> — properties table <i>name</i> — property name <i>value</i> — value of <i>name</i>
Return Values	TRUE on success, FALSE otherwise.
Diagnostics	If TipcPropertiesSet fails, it returns FALSE and sets the global SmartSockets error code to: • T_ERR_NULL_PTR — <i>props</i> was null or <i>name</i> was null
Description	TipcPropertiesSet associates a value with a property name. If <i>value</i> is null then this property is deleted from the table. This effectively disables the property.
Caution	None
See Also	TipcPropertiesCreate, TipcPropertiesGet
Examples	This example sets the values in a property table: <pre>T_IPC_PROPERTIES props; T_BOOL status; status = TipcPropertiesSet(props, "prop1", "val1"); if (!status) { return; /* error */ }</pre>

TipcPropertiesTraverse

Name	TipcPropertiesTraverse — traverse the specified property list
Synopsis	<pre>T_STR TipcPropertiesTraverse(props, traverse_func, traverse_arg) T_IPC_PROPERTIES props; T_IPC_PROPERTY_TRAV_FUNC traverse_func; T_PTR traverse_arg;</pre>
Arguments	<i>props</i> — properties table <i>traverse_func</i> — function to call once for each entry in properties table <i>traverse_arg</i> — argument to pass to <i>traverse_func</i>
Return Values	TipcPropertiesTraverse returns the first non-null return value from <i>traverse_func</i> , or NULL if <i>traverse_func</i> always returns NULL.
Diagnostics	If TipcPropertiesTraverse fails or traverses to the end of the properties table, it returns NULL and sets the global SmartSockets error code to one of: • T_ERR_NULL_PTR — <i>props</i> was null • T_ERR_END_OF_TRAVERSAL — all entries in table were traversed
Description	TipcPropertiesTraverse traverses all the entries in the properties table and calls <i>traverse_func</i> for each entry. <i>traverse_func</i> is called with three arguments: <i>key</i> , <i>value</i> and <i>traverse_arg</i> . When <i>traverse_func</i> returns a null value then TipcPropertiesTraverse returns with that value.
Caution	Do not add or delete entries with <i>traverse_func</i> . The ordering of the hash entries is not guaranteed.
See Also	TipcPropertiesCreate, TipcPropertiesGet, TipcPropertiesSet
Examples	This example traverses a properties table: <pre>T_IPC_PROPERTIES props; T_PTR traverse_func(key, value, arg) T_PTR key; T_PTR value; T_PTR arg; { TutOut("key = %s, value = %s\n", (T_STR)key, (T_STR)value); } TipcPropertiesTraverse(props, traverse_func, NULL);</pre>

Chapter 13 **TipcSrv***

This chapter covers the TipcSrv API functions that handle single server connections. For information on the TipcSrvConn API functions that handle multiple connections, see Chapter 14, TipcSrvConn*. This chapter covers these TipcSrv API functions:

- *TipcSrvBufferGetReadSize*
- *TipcSrvBufferGetWriteSize*
- *TipcSrvCheck*
- *TipcSrvCreate*
- *TipcSrvCreateCbCreate*
- *TipcSrvCreateCbLookup*
- *TipcSrvDefaultCbCreate*
- *TipcSrvDefaultCbCreate*
- *TipcSrvDefaultCbLookup*
- *TipcSrvDestroy*
- *TipcSrvDestroyCbCreate*
- *TipcSrvDestroyCbLookup*
- *TipcSrvErrorCbCreate*
- *TipcSrvErrorCbLookup*
- *TipcSrvFlush*
- *TipcSrvGetArch*
- *TipcSrvGetAutoFlushSize*
- *TipcSrvGetBlockMode*
- *TipcSrvGetCurrentConnectedLcn*
- *TipcSrvGetConnStatus*
- *TipcSrvGetGmdMaxSize*
- *TipcSrvGetGmdNumPending*
- *TipcSrvGetNode*

- *TipcSrvGetNumQueued*
- *TipcSrvGetPid*
- *TipcSrvGetSocket*
- *TipcSrvGetTimeout*
- *TipcSrvGetUniqueSubject*
- *TipcSrvGetUser*
- *TipcSrvGetXtSource*
- *TipcSrvGmdFileCreate*
- *TipcSrvGmdFileDelete*
- *TipcSrvGmdMsgDelete*
- *TipcSrvGmdMsgServerDelete*
- *TipcSrvGmdMsgStatus*
- *TipcSrvIsRunning*
- *TipcSrvKeepAlive*
- *TipcSrvLock*
- *TipcSrvLogAddMt*
- *TipcSrvLogRemoveMt*
- *TipcSrvMainLoop*
- *TipcSrvMsgInsert*
- *TipcSrvMsgNext*
- *TipcSrvMsgProcess*
- *TipcSrvMsgSearch*
- *TipcSrvMsgSearchType*
- *TipcSrvMsgSend*
- *TipcSrvMsgSendRpc*
- *TipcSrvMsgWrite*
- *TipcSrvMsgWriteVa*
- *TipcSrvPrint*
- *TipcSrvProcessCbCreate*
- *TipcSrvProcessCbLookup*

- *TipcSrvQueueCbCreate*
- *TipcSrvQueueCbLookup*
- *TipcSrvRead*
- *TipcSrvReadCbCreate*
- *TipcSrvReadCbLookup*
- *TipcSrvSetAutoFlushSize*
- *TipcSrvSetBlockMode*
- *TipcSrvSetCredentials*
- *TipcSrvSetGmdMaxSize*
- *TipcSrvSetSocket*
- *TipcSrvSetTimeout*
- *TipcSrvSetUsernamePassword*
- *TipcSrvStdSubjectSetSubscribe*
- *TipcSrvStop*
- *TipcSrvSubjectCbCreate*
- *TipcSrvSubjectCbDestroyAll*
- *TipcSrvSubjectCbLookup*
- *TipcSrvSubjectDefaultCbCreate*
- *TipcSrvSubjectDefaultCbLookup*
- *TipcSrvSubjectGetSubscribe*
- *TipcSrvSubjectGetSubscribeLb*
- *TipcSrvSubjectGmdInit*
- *TipcSrvSubjectLbInit*
- *TipcSrvSubjectSetSubscribe*
- *TipcSrvSubjectSetSubscribeEx*
- *TipcSrvSubjectSetSubscribeLb*
- *TipcSrvSubjectTraverseSubscribe*
- *TipcSrvTrafficGetBytesRecv8*
- *TipcSrvTrafficGetBytesSent8*
- *TipcSrvTrafficGetMsgsRecv8*

- *TpcSrvTrafficGetMsgsSent8*
- *TpcSrvTraverseCbCreate*
- *TpcSrvTraverseCbLookup*
- *TpcSrvUnlock*
- *TpcSrvWriteCbCreate*
- *TpcSrvWriteCbLookup*

TipcSrvBufferGetReadSize

Name	TipcSrvBufferGetReadSize — get the total number of bytes in a connection's read buffer
Synopsis	<pre>T_BOOL TipcSrvBufferGetReadSize(<i>read_size_return</i>) T_INT4 *<i>read_size_return</i>;</pre>
Arguments	<i>read_size_return</i> — returned number of bytes
Return Values	TRUE if the number of bytes was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcSrvBufferGetReadSize fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>read_size_return</i> was null
Description	TipcSrvBufferGetReadSize retrieves the total number of bytes in a connection's read buffer.
See Also	TipcSrvBufferGetWriteSize
Examples	This example retrieves the number of bytes in a connection's read buffer: <pre>T_INT4 read_size; . . if (!TipcSrvBufferGetReadSize(&read_size)) { return; /* error */ }</pre>

TipcSrvBufferGetWriteSize

Name	TipcSrvBufferGetWriteSize — get the total number of bytes in a connection’s write buffer
Synopsis	<pre>T_BOOL TipcSrvBufferGetWriteSize(write_size_return) T_INT4 *write_size_return;</pre>
Arguments	<i>write_size_return</i> — returned number of bytes
Return Values	TRUE if the number of bytes was sucessfully retrieved, FALSE otherwise.
Diagnostics	If TipcSrvBufferGetWriteSize fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>write_size_return</i> was null
Description	TipcSrvBufferGetWriteSize retrieves the total number of bytes in a connection’s write buffer.
See Also	TipcSrvBufferGetReadSize
Examples	This example retrieves the number of bytes in a connection’s write buffer: <pre>T_INT4 write_size; . . if (!TipcSrvBufferGetWriteSize(&write_size)) { return; /* error */ }</pre>

TipcSrvCheck

Name	TipcSrvCheck — check if data can be read from or written to connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvCheck(<i>check_mode</i>, <i>timeout</i>) T_IO_CHECK_MODE <i>check_mode</i>; T_REAL8 <i>timeout</i>;</pre>
Arguments	<i>check_mode</i> — how to check RTserver availability <i>timeout</i> — maximum number of seconds to wait for availability
Return Values	TRUE if data can be read or written, FALSE if <i>timeout</i> reached or an error occurred.
Diagnostics	If TipcSrvCheck fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnCheck
Description	TipcSrvCheck checks the connection to RTserver by calling TipcConnCheck. The connection can be checked if data can be read (<i>check_mode</i> is T_IO_CHECK_READ) or if data can be written (<i>check_mode</i> is T_IO_CHECK_WRITE). See TipcConnCheck for more information on checking connections. TipcSrvCheck may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvCheck sleeps for <i>timeout</i> seconds and returns FALSE. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	TipcSrvCheck is a low-level function that is normally not used directly by developers. TipcConnCheck is normally called only by TipcSrvFlush and TipcSrvRead.
See Also	TipcConnCheck, TipcSrvRead
Examples	This example uses TipcSrvCheck to check if data is available within 3.4 seconds for reading from the connection to RTserver: <pre>if (!TipcSrvCheck(T_IO_CHECK_READ, 3.4)) { return; /* error */ }</pre>

TipcSrvCreate

Name	TipcSrvCreate — create the connection to RTserver										
Synopsis	<pre>T_BOOL TipcSrvCreate(create_status) T_IPC_SRV_CONN_STATUS create_status;</pre>										
Arguments	<i>create_status</i> — whether to create a warm connection (use T_IPC_SRV_CONN_WARM) or a full connection (use T_IPC_SRV_CONN_FULL)										
Return Values	TRUE if the connection to RTserver was successfully created, FALSE otherwise.										
Diagnostics	If TipcSrvCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>create_status</i> was not T_IPC_SRV_CONN_WARM or T_IPC_SRV_CONN_FULL • T_ERR_SRV_ALREADY_CONNECTED — the process already has a connection to RTserver • T_ERR_SRV_ACCESS_DENIED — access was denied because another RTclient or RTserver has the same value for the option Unique_Subject • T_ERR_SRV_NOT_CONNECTED — a connection to an RTserver could not be created										
Description	TipcSrvCreate creates the connection to RTserver. The RTclient becomes part of the project named in the option Project, can subscribe to subjects, can publish (send) messages to RTserver, and receive messages from RTserver. Each project is self-contained; messages cannot be sent between projects. When the connection to RTserver is created, the RTclient immediately starts subscribing to the subject named in the option Unique_Subject. The RTclient can call TipcSrvSubjectSetSubscribe to start subscribing to other subjects. TipcSrvCreate uses several RTclient options to control the creation of the connection to RTserver: <table><tr><td>Default_Protocols</td><td>specifies the list of IPC protocols to try if no protocol is specified</td></tr><tr><td>Server_Disconnect_Mode</td><td>specifies the action RTserver should take when RTclient disconnects</td></tr><tr><td>Server_Names</td><td>specifies the list of logical connection names to use to find RTserver</td></tr><tr><td>Server_Start_Delay</td><td>specifies the number of seconds to sleep between tries</td></tr><tr><td>Server_Start_Max_Tries</td><td>specifies the number of times to try to connect</td></tr></table>	Default_Protocols	specifies the list of IPC protocols to try if no protocol is specified	Server_Disconnect_Mode	specifies the action RTserver should take when RTclient disconnects	Server_Names	specifies the list of logical connection names to use to find RTserver	Server_Start_Delay	specifies the number of seconds to sleep between tries	Server_Start_Max_Tries	specifies the number of times to try to connect
Default_Protocols	specifies the list of IPC protocols to try if no protocol is specified										
Server_Disconnect_Mode	specifies the action RTserver should take when RTclient disconnects										
Server_Names	specifies the list of logical connection names to use to find RTserver										
Server_Start_Delay	specifies the number of seconds to sleep between tries										
Server_Start_Max_Tries	specifies the number of times to try to connect										

<code>Server_Start_Timeout</code>	specifies the number of seconds to wait for RTserver to initialize
<code>Udp_Broadcast_Timeout</code>	specifies the number of seconds to wait for broadcast replies

In addition to these options, the boolean option `Server_Auto_Connect` is related to `TipcSrvCreate`. An RTclient can have a warm connection to RTserver, which is a subset of a full connection to RTserver. Most of the other `TipcSrv` functions, such as `TipcSrvMsgSend`, call `TipcSrvCreate` if `Server_Auto_Connect` is set to `TRUE` and the RTclient does not have a full connection to RTserver. See `TipcSrvDestroy` for more information on warm connections to RTserver.

Logical Connection Names in `Server_Names`

The most important option is `Server_Names`, which is used to find and start RTserver. `Server_Names` is a list of logical connection names. Each logical connection name has the form *protocol:node:address*, which can be shortened to *protocol:node*, *protocol*, or *node*, and is specified this way:

protocol refers to an IPC protocol type. The valid values for *protocol* are `local`, `tcp` and `udp_broadcast`. If *protocol* is omitted from the connection name, then all protocols listed in `Default_Protocols` are tried.

The `udp_broadcast` protocol is only used to find RTserver and not to start RTserver or send or receive messages. When the `udp_broadcast` protocol is used, a packet is broadcast to all nodes on the network to attempt to find an RTserver. If an RTserver receives the broadcast packet, it responds to the request with a list of non-broadcast connection names for RTclient to use to find that RTserver. RTclient waits up to `Udp_Broadcast_Timeout` seconds for RTserver to respond to its broadcast, and then connects to the RTserver that responds first.

node refers to a computer node name. The special value `_node` can be used for *node* to indicate the name of the current node; this is useful for generalizing a connection name to work on any computer.

address refers to a protocol-specific IPC location, such as a `tcp` port number. If *address* is omitted from the connection name, a protocol-specific default *address* is used. See the *TIBCO SmartSockets User's Guide* for a list of these defaults.

Using a Start Prefix

Each logical connection name can also have a start prefix at the front, which must be separated from the name with a colon:

start_prefix:protocol:node:address

The value for *start_prefix* controls whether and when the RTclient tries to start RTserver:

- start_always* specifies that the RTclient always tries to start RTserver if it cannot create the connection. This is the default. If no start prefix is specified, an implicit prefix of *start_always* is used (unless *protocol* is *local* and *node* is not the name of the current node, in which case the default is *start_never*).
- start_on_demand* specifies that the RTclient only tries to start RTserver if the RTclient has tried all names in *Server_Names* at least once; this is useful for only starting RTserver if an existing one cannot be found.
- start_never* specifies that the RTclient never tries to start RTserver. If *protocol* is *local* and *node* is not the name of the current node, an implicit prefix of *start_never* is used because the local protocol cannot connect to a remote RTserver.

Traversing Server_Names

A client traverses the list of names in *Server_Names* at most *Server_Start_Max_Tries*. Between each traversal the client sleeps for *Server_Start_Delay* seconds. Each time an RTclient tries to start RTserver, it waits for up to *Server_Start_Timeout* seconds for that RTserver to finish initializing. For each entry in *Server_Names*, a client tries to connect, then possibly try to start RTserver (depending on what start prefix is used), then try to connect again.

This example shows the algorithm used by *TipcSrvCreate*:

```
randomize Server_Names list if _random is used

for (each $try from 1 to Server_Start_Max_Tries) {

    for (each $name in Server_Names) {
        connect to RTserver using $name
        if (connection could be created) {
            if (no connection previously existed) {
                resend old GMD messages
            }
            execute server create callbacks
            return
        }
    }
}
```



```

else {
    if ($name has the start prefix "start_always"
        or $name has no start prefix
        or ($name has the start prefix "start_on_demand"
            and $try > 1)) {
        start RTserver with command "rtserver -node node"
        wait up to Server_Start_Timeout seconds for RTserver to init
        connect to RTserver using $name
        if (connection could be created) {
            if (no connection previously existed) {
                resend old GMD messages
            }
            execute server create callbacks
            return
        }
    } /* if we should try to start RTserver */
} /* if no connection */
} /* for each name */

sleep for Server_Start_Delay seconds
} /* for each try */

```



The `start_on_demand` start prefix only starts RTserver if the option `Server_Start_Max_Tries` is set to a value larger than 1 (the default).

RTclient Options

The RTclient options (Project, Server_Names, and so on) are only accessed when the RTclient creates a connection to RTserver (see below for special handling for Project, Unique_Subject, and Default_Subject_Prefix). If one of these options is changed, it has no effect until the next time the RTclient creates a connection to RTserver. One possible way of doing this is to destroy the connection to RTserver (but keep it warm) and then create a new connection to RTserver using the warm information. This code can be used to change an option like Project and have it take effect immediately:

```

/* destroy existing connection to RTserver, but keep it warm */
if (!TipcSrvDestroy(T_IPC_SRV_CONN_WARM)) {
    return; /* error */
}

/* change Project */
TutCommandParseStr("setopt project new_project");

/* create a new connection to RTserver from warm info */
if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}

```

The Project option is set as read-only while RTclient has a full connection to RTserver. This prevents confusion in case this option is changed after the full connection is created.

Because the Unique_Subject and Default_Subject_Prefix options may be used each time RTclient publishes a message, these options are set as read-only while RTclient has a warm or full connection to RTserver. This prevents misconfiguration during normal publish-subscribe operation. For a change in Unique_Subject or Default_Subject_Prefix to take effect, the connection to RTserver must be fully destroyed. (No warm connection can be left.)

GMD Handling

If the RTclient had no connection at all to RTserver, the TipcSrvCreate calls TipcSrvGmdResend to resend any old messages for GMD. This is needed to preserve message sequence number order, even for a warm connection. TipcSrvCreate does not create any directories for GMD if no resending is needed. This prevents large numbers of unused directories from being created for large projects.

Creating a Full Connection

If a full connection to RTserver is created, some one-time handshaking between RTclient and RTserver takes place. RTclient sends a CONNECT_CALL message to RTserver and RTserver responds with a CONNECT_RESULT message.

Callbacks

TipcSrvCreate also creates many callbacks, several of which can be destroyed by the user if the user wants to replace the default functionality:

- TipcSrvErrorCbCreate is called with function TipcCbSrvError and argument NULL to create an error callback.
- TipcSrvProcessCbCreate is called with message type CONTROL, function TipcCbSrvProcessControl, and argument NULL to create a process callback.
- TipcSrvProcessCbCreate is called with message type GMD_FAILURE, function TipcCbSrvProcessGmdFailure, and argument NULL to create a process callback.
- Several connection process callbacks are created for TipcMon* monitoring.
- Several option change callbacks are created to handle the Log_* and Server_* options.

Caution An RTclient cannot have a connection to more than one RTserver at a time unless using the TipcSrvConn API functions that handle multiple connections.

See Also TipcSrvDestroy, TipcSrvGetConnStatus

Examples This example sets the options Project and Server_Names, then connects to RTserver:

```
TutCommandParseStr("setopt project demo");
TutCommandParseStr("setopt server_names tcp:moe");

if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}
```

TipcSrvCreateCbCreate

Name	TipcSrvCreateCbCreate — create a server create callback
Synopsis	<pre>T_CB TipcSrvCreateCbCreate(<i>func</i>, <i>arg</i>) T_IPC_SRV_CREATE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvCreateCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcSrvCreateCbCreate creates a server create callback. These callbacks are called when an RTclient creates a warm or full connection to RTserver. See the <i>TIBCO SmartSockets Utilities</i> reference for more information on callbacks. Server destroy callbacks are the opposite of server create callbacks. Server destroy callbacks can be used to perform processing when RTclient destroys its connection to RTserver. See TipcSrvDestroyCbCreate for more information on server destroy callbacks. When the callback function is called, the second function argument contains the callback data (C type T_IPC_SRV_CREATE_CB_DATA). The fields <i>old_conn_status</i> and <i>new_conn_status</i> (both have C type T_IPC_SRV_CONN_STATUS) in this callback data can be used to determine the previous and current status of the connection to RTserver. The field <i>old_conn_status</i> contains either the value T_IPC_SRV_CONN_NONE or the value T_IPC_SRV_CONN_WARM. The field <i>new_conn_status</i> contains the value T_IPC_SRV_CONN_WARM or the value T_IPC_SRV_CONN_FULL. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcSrvCreateCbLookup, TipcSrvDestroyCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy

Examples This example shows an application that creates a server create callback which is used to print a message when a new server connection is made:

```
#include <rtworks/ipc.h>

/* =====
/* ..my_server_create_cb -- server create callback */
void T_ENTRY my_server_create_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_SRV_CREATE_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    TutOut("Server connection created. \n");
} /* my_server_create_cb */

/* =====
/* ..main -- main program */
int main(int argc, char **argv)
{
    /* create a callback to be called whenever we connect */
    /* to RTserver */
    if (TipcSrvCreateCbCreate(my_server_create_cb, NULL) == NULL) {
        return; /* error */
    }
} /* main */
```

TipcSrvCreateCbLookup

Name	TipcSrvCreateCbLookup — look up a server create callback
Synopsis	<pre>T_CB TipcSrvCreateCbLookup(<i>func</i>, <i>arg</i>) T_IPC_SRV_CREATE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcSrvCreateCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcSrvCreateCbLookup looks up a server create callback with the desired function and argument. These callbacks are called when an RTclient creates a warm or full connection to RTserver. See TipcSrvCreateCbCreate for more information on server create callbacks.
Caution	None
See Also	TipcSrvCreateCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up a server create callback: <pre>T_CB cb; cb = TipcSrvCreateCbLookup(my_server_create_cb, NULL); if (cb == NULL) { return; /* error */ }</pre>

TIPC_SrvDefaultCbCreate

Name	TIPC_SrvDefaultCbCreate — create a default callback in the connection to RTserver
Synopsis	<pre>T_CB TIPC_SrvDefaultCbCreate(<i>func</i>, <i>arg</i>) T_IPC_CONN_DEFAULT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TIPC_SrvDefaultCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TIPC_ConnDefaultCbCreate
Description	TIPC_SrvDefaultCbCreate creates a default callback in the connection to RTserver by calling TIPC_ConnDefaultCbCreate. These callbacks are called by TIPC_SrvMsgProcess when there are no process callbacks for the type of message being processed. See TIPC_ConnDefaultCbCreate for more information on default callbacks. TIPC_SrvDefaultCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TIPC_SrvDefaultCbCreate creates a default callback in the warm connection. See TIPC_SrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TIPC_ConnDefaultCbCreate, TIPC_SrvMsgProcess, TIPC_SrvDefaultCbLookup

Examples This example creates a default callback which warns that an unexpected message has arrived from the connection to RTserver:

```
void T_ENTRY my_srv_default_cb(conn, data, arg)
T_IPC_CONN conn; /* connection to RTserver */
T_IPC_CONN_DEFAULT_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    if (!TipcMsgPrintError(data->msg)) {
        return; /* error */
    }
} /* my_srv_default_cb */

/* ===== */
/*...code from calling function is shown below */

/* create default callback */
if (TipcSrvDefaultCbCreate(my_srv_default_cb, NULL) == NULL) {
    return; /* error */
}
```


TipcSrvDefaultCbLookup

Name	TipcSrvDefaultCbLookup — look up a default callback in the connection to RTserver
Synopsis	<pre>T_CB_TipcSrvDefaultCbLookup(<i>func</i>, <i>arg</i>) T_IPC_CONN_DEFAULT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcSrvDefaultCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnDefaultCbLookup
Description	TipcSrvDefaultCbLookup looks up a default callback with the desired function and argument in the connection to RTserver by calling TipcConnDefaultCbLookup. These callbacks are called by TipcSrvMsgProcess when there are no process callbacks for the type of message being processed. See TipcConnDefaultCbCreate for more information on default callbacks. TipcSrvDefaultCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvDefaultCbLookup looks up a default callback in the warm connection. See TipcSrvCreate for information on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcSrvDefaultCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This looks up and destroys a default callback in the connection to RTserver:

```
cb = TipcSrvDefaultCbLookup(my_srv_default_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
if (!TutCbDestroy(cb)) {
    return; /* error */
}
```

TipcSrvDestroy

Name	TipcSrvDestroy — destroy the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvDestroy(<i>destroy_status</i>) T_IPC_SRV_CONN_STATUS <i>destroy_status</i>;</pre>
Arguments	<i>destroy_status</i> — whether to keep a warm connection to RTserver (use T_IPC_SRV_CONN_WARM) or fully destroy the connection to RTserver (use T_IPC_SRV_CONN_NONE)
Return Values	TRUE if the connection to RTserver was successfully destroyed, FALSE otherwise.
Diagnostics	If TipcSrvDestroy fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>destroy_status</i> was not T_IPC_SRV_CONN_WARM or T_IPC_SRV_CONN_NONE • T_ERR_SRV_NOT_CONNECTED — <i>destroy_status</i> was T_IPC_SRV_CONN_WARM but there was no full connection to RTserver, or <i>destroy_status</i> was T_IPC_SRV_CONN_NONE and there was no (full or warm) connection to RTserver
Description	TipcSrvDestroy destroys the connection to RTserver. There are two ways to destroy the connection to RTserver. A full destroy removes all of RTclient's RTserver-related information. Once this has occurred, RTclient can continue as if it had never been connected at all to RTserver. The other mode of TipcSrvDestroy is a warm destroy. A warm destroy keeps as much RTserver-related information as possible. Once this has occurred, RTclient can easily reconnect to RTserver in the future and pick up where it left off. After a warm destroy, RTclient can even partially operate as if it still had a connection to RTserver. Messages can be sent, and they are buffered until a full connection to RTserver is created again. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver. If RTclient has a full connection to RTserver when TipcSrvDestroy is called and the connection error callbacks aren't being called (which implies the connection is unusable), then TipcSrvDestroy first tries to send a DISCONNECT message to RTserver. This DISCONNECT message has one field, an integer based on the value of the option Server_Disconnect_Mode. The DISCONNECT message is not necessary for RTserver to detect that RTclient has destroyed its connection to RTserver, but the DISCONNECT message does provide a more graceful disconnect and also allows RTclient to change Server_Disconnect_Mode dynamically before calling TipcSrvDestroy.

TipcSrvDestroy does not destroy any message types, options, server create callbacks, or server destroy callbacks. These entities only need to be created once, not every time TipcSrvCreate is called.

Destroy Status T_IPC_SRV_CONN_WARM

If *destroy_status* is T_IPC_SRV_CONN_WARM, then TipcSrvDestroy first closes the socket in the connection to RTserver. (As soon as the socket is closed, RTserver destroys its own connection to RTclient.) TipcSrvDestroy then removes all buffered incoming messages from the connection read buffer and all buffered outgoing messages from the connection write buffer. All messages in the connection message queue are left in place, however. TipcSrvDestroy next calls TipcSrvGmdResend to resend any messages for GMD; this primes the warm connection with GMD messages so that sequence number order is preserved. The server destroy callbacks are then called.

Destroy Status T_IPC_SRV_NONE

If *destroy_status* is T_IPC_SRV_NONE, then TipcSrvDestroy removes all local subject information, destroys all internal callbacks (for options like Server_Read_Timeout and Log_In_Data), destroys the connection, and finally calls the server destroy callbacks.

RTclient Options

The RTclient options (Project, Server_Names, and so on) are only accessed when the RTclient creates a connection to RTserver (see below for special handling for Project, Unique_Subject, and Default_Subject_Prefix). If one of these options is changed, it has no effect until the next time the RTclient creates a connection to RTserver. One possible way of doing this is to destroy the connection to RTserver (but keep it warm) and then create a new connection to RTserver using the warm information. This code can be used to change an option like Project and have it take effect immediately:

```
/* destroy existing connection to RTserver, but keep it warm */
if (!TipcSrvDestroy(T_IPC_SRV_CONN_WARM)) {
    return; /* error */
}

/* change project */
TutCommandParseStr("setopt project new_project");

/* create a new connection to RTserver from warm info */
if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}
```

The Project option is set as read-only while RTclient has a full connection to RTserver. This prevents confusion from resulting if this option is changed after the full connection is created.

Because the Unique_Subject and Default_Subject_Prefix options may be used each time RTclient publishes a message, these options are set as read-only while RTclient has a warm or full connection to RTserver. This prevents misconfiguration during normal publish-subscribe operation. For a change in Unique_Subject or Default_Subject_Prefix to take effect, the connection to RTserver must be fully destroyed (no warm connection can be left) and recreated.

Caution An RTclient using the TCP/IP protocol to connect to RTserver may lose outgoing messages if the process terminates without calling TipcSrvDestroy. TCP/IP's SO_LINGER option, which preserves data, is ignored when closing a socket that has non-blocking I/O enabled. While data loss rarely occurs on UNIX and OpenVMS, it can happen very frequently on Windows. TipcSrvDestroy sets the block mode of the connection to TRUE before closing the connection's socket, which forces the operating system to deliver all flushed outgoing messages.

Messages that have been sent to RTserver but not flushed may be lost after a call to TipcSrvDestroy.

Use TipcSrvDestroy(T_IPC_SRV_CONN_NONE) with caution from a connection callback such as a connection process callback. Because TipcSrvDestroy(T_IPC_SRV_CONN_NONE) completely destroys the connection, the program crashes if control from the callback is returned to a function such as TipcSrvMainLoop.

See Also TipcSrvCreate, TipcSrvGetConnStatus

Examples This example creates a connection to RTserver, sends a message, and then fully destroys the connection to RTserver with TipcSrvDestroy:

```
T_IPC_MT mt;

/* Create a connection to RTserver. */
if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

/* Send a message and make sure it gets flushed out. */
if (!TipcSrvMsgWrite("/system/admin", mt, TRUE,
                    T_IPC_FT_STR, "speed_limit",
                    T_IPC_FT_REAL8, 65.0,
                    NULL)) {
    return; /* error */
}

if (!TipcSrvFlush()) {
    return; /* error */
}

/* Fully destroy the connection to RTserver. */
if (!TipcSrvDestroy(T_IPC_SRV_CONN_NONE)) {
    return; /* error */
}
```

TipcSrvDestroyCbCreate

Name	TipcSrvDestroyCbCreate — create a server destroy callback
Synopsis	<pre>T_CB TipcSrvDestroyCbCreate(<i>func</i>, <i>arg</i>) T_IPC_SRV_DESTROY_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvDestroyCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcSrvDestroyCbCreate creates a server destroy callback. These callbacks are called when an RTclient destroys its connection to RTserver. Server destroy callbacks are useful for breaking any dependencies that RTclient has on being connected to RTserver. See the <i>TIBCO SmartSockets Utilities</i> reference for more information on callbacks. Server create callbacks are the opposite of server destroy callbacks. Server create callbacks can be used to perform processing when an RTclient creates a connection to RTserver. See TipcSrvCreateCbCreate for more information on server create callbacks. When the callback function is called, the second function argument contains the callback data (C type T_IPC_SRV_DESTROY_CB_DATA). The fields <i>old_conn_status</i> and <i>new_conn_status</i> (both have C type T_IPC_SRV_CONN_STATUS) in this callback data can be used to determine the previous and current status of the connection to RTserver. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	The first argument passed to <i>func</i> is the current connection to RTserver. This argument is null if there is currently no connection to RTserver. A server destroy callback should not call any of the TipcSrv* functions that attempt to automatically connect to RTserver, as this would cause the server destroy callback to reconnect to RTserver. The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.

See Also TpcSrvCreateCbCreate, TpcSrvDestroyCbLookup; see the *TIBCO SmartSockets Utilities* for information on TutCbDestroy.

Examples This example shows how to use server destroy (and server create) callbacks to mix Motif and a connection to RTserver:

```
/*..my_xt_func -- read and process data from RTserver */
void my_xt_func(client_data, source, id)
XtPointer client_data; /* not used */
int *source; /* not used */
XtInputId *id; /* not used */
{
    TpcSrvMainLoop(0.0);
} /* my_xt_func */

/*..my_server_destroy_cb -- unhook RTserver connection from Motif */
void T_ENTRY my_server_destroy_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_SRV_DESTROY_CB_DATA data;
T_CB_ARG arg; /* really (XtInputId *) */
{
    XtInputId *id_ptr = (XtInputId *)arg;

    if (*id_ptr != NULL) {
        XtRemoveInput(*id_ptr);

        /* clear out id_ptr so that we don't try to remove the */
        /* same Xt ID twice */
        *id_ptr = NULL;
    }
} /* my_server_destroy_cb */

/*..my_server_create_cb -- hook RTserver connection into Motif */
void T_ENTRY my_server_create_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_SRV_CREATE_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_INT4 source;
    static XtInputId my_id; /* permanent storage */

    if (!TpcSrvGetXtSource(&source)) {
        return; /* error */
    }
}
```



```

        /* create Xt input callback when data available from the */
        /* connection to RTserver */
        my_id = XtAppAddInput(app_context,
                               (int)source,
#ifdef T_OS_VMS
                               NULL,
#else
                               (XtPointer)XtInputReadMask,
#endif
                               my_xt_func,
                               NULL);

        /* create server destroy callback to remove Xt input callback */
        if (TipcSrvDestroyCbCreate(my_server_destroy_cb, &my_id) == NULL)
        {
            return; /* error */
        }

    } /* my_server_create_cb */

    /* ===== */
    /*...code from main program is below */

    /* NOTE: this can only be used if Motif is initialized before */
    /* the process will connect to RTserver */

    if (TipcSrvCreateCbCreate(my_server_create_cb, NULL) == NULL) {
        return; /* error */
    }

```

TipcSrvDestroyCbLookup

Name	TipcSrvDestroyCbLookup — look up a server destroy callback
Synopsis	<pre>T_CB TipcSrvDestroyCbLookup(<i>func</i>, <i>arg</i>) T_IPC_SRV_DESTROY_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcSrvDestroyCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcSrvDestroyCbLookup looks up a server destroy callback with the desired function and argument. These callbacks are called when RTclient destroys its connection to RTserver. See TipcSrvDestroyCbCreate for more information on server destroy callbacks.
Caution	None
See Also	TipcSrvDestroyCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up a server destroy callback: <pre>T_CB cb; cb = TipcSrvDestroyCbLookup(my_server_destroy_cb, NULL); if (cb == NULL) { return; /* error */ }</pre>

TipcSrvErrorCbCreate

Name	TipcSrvErrorCbCreate — create an error callback in the connection to RTserver
Synopsis	<pre>T_CB TipcSrvErrorCbCreate(<i>func</i>, <i>arg</i>) T_IPC_CONN_ERROR_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvErrorCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnErrorCbCreate
Description	TipcSrvErrorCbCreate creates an error callback in the connection to RTserver by calling TipcConnErrorCbCreate. These callbacks are called when there is an unrecoverable error on the connection to RTserver. See TipcConnErrorCbCreate for more information on error callbacks. TipcSrvErrorCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvErrorCbCreate creates an error callback in the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver. When an RTclient creates a connection to RTserver with TipcSrvCreate, an error callback is created in the connection with function TipcCbSrvError, argument null, and priority 0. TipcCbSrvError restarts RTserver, reconnects to RTserver, starts subscribing to the necessary subjects again, and starts watching the necessary subjects again. If an RTclient needs to take any additional recovery steps, such as checking for application-specific hardware failures, it can create an error callback with a lower priority (which is called after TipcCbSrvError).
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcCbSrvError, TipcConnErrorCbCreate, TipcSrvErrorCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example creates an error callback for the connection to RTserver that is called before the error callback TipcCbSrvError:

```
/* ===== */
/*..my_srv_error_cb -- example server error callback func */
void T_ENTRY my_srv_error_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_ERROR_CB_DATA data;
T_CB_ARG arg;
{
    /* perform necessary error recovery here */
} /* my_srv_error_cb */

/*...fragment that uses above function is shown below */

T_CB cb;
T_CB_PRIORITY priority;

cb = TipcSrvErrorCbLookup(TipcCbSrvError, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbGetPriority(cb, &priority)) {
    return; /* error */
}

cb = TipcSrvErrorCbCreate(my_srv_error_cb, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbSetPriority(cb, priority + 1)) {
    return; /* error */
}
```

TipcSrvErrorCbLookup

Name	TipcSrvErrorCbLookup — look up an error callback in the connection to RTserver
Synopsis	<pre>T_CB TipcSrvErrorCbLookup(<i>func</i>, <i>arg</i>) T_IPC_CONN_ERROR_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcSrvErrorCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnErrorCbLookup
Description	TipcSrvErrorCbLookup looks up an error callback with the desired function and argument in the connection to RTserver by calling TipcConnErrorCbLookup. These callbacks are called when there is an unrecoverable error on the connection to RTserver. See TipcConnErrorCbCreate more information on error callbacks. TipcSrvErrorCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvErrorCbLookup looks up an error callback in the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcSrvErrorCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up an error callback in the connection to RTserver: <pre>T_CB cb; cb = TipcSrvErrorCbLookup(my_srv_error_cb, NULL); if (cb == NULL) { return; /* error */ }</pre>

TipcSrvFlush

Name	TipcSrvFlush — flush buffered outgoing messages on the connection to RTserver
Synopsis	T_BOOL TipcSrvFlush()
Arguments	None
Return Values	TRUE if any data was successfully flushed to the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvFlush fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnFlush
Description	TipcSrvFlush flushes buffered outgoing messages on the connection to RTserver by calling TipcConnFlush. See TipcConnFlush for more information on flushing outgoing messages. TipcSrvFlush may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvFlush returns FALSE immediately. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnFlush, TipcSrvMsgSend
Examples	This example shows how to use TipcSrvFlush to flush a message to the connection to RTserver: <pre>if (!TipcSrvMsgSend(msg, TRUE)) { return; /* error */ } if (!TipcSrvFlush()) { return; /* error */ }</pre>

TipcSrvGetArch

Name	TipcSrvGetArch — get the architecture name of the connected RTserver
Synopsis	<pre>T_BOOL TipcSrvGetArch(<i>arch_return</i>) T_STR *<i>arch_return</i>;</pre>
Arguments	<i>arch_return</i> — storage for RTserver architecture name
Return Values	TRUE if the architecture name was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetArch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>arch_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetArch
Description	TipcSrvGetArch gets the architecture name of the RTserver this RTclient is connected to by calling TipcConnGetArch. The RTserver architecture name can be useful for diagnostic purposes. The architecture name is the SmartSockets architecture, using the format: <i>Machine_OperatingSystem</i> where: <i>Machine</i> is the hardware name of the system. <i>OperatingSystem</i> is the operating system used. For example: sun4_solaris This value is stored in the environment variable RTARCH. There is some overlap between TipcSrvGetArch and TipcMonServerGeneralPoll, but TipcSrvGetArch is easier to use if only the architecture name is needed instead of all RTserver general information. TipcSrvGetArch may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetArch does not use the warm connection, because the warm connection does not have a peer process with an architecture name. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.

Caution If TipcSrvGetArch returns FALSE, it does not store a value in *arch_return*.

The value stored in *arch_return* points directly into an internal data structure and should not be modified.

See Also TipcConnGetArch, TipcMonServerGeneralPoll, TipcSrvGetNode, TipcSrvGetPid, TipcSrvGetUniqueSubject, TipcSrvGetUser

Examples This example prints the architecture name of the connected RTserver:

```
T_STR arch;

if (!TipcSrvGetArch(&arch)) {
    return; /* error */
}

TutOut("RTserver architecture name is <%s>.\n", arch);
```


TipcSrvGetAutoFlushSize

Name	TipcSrvGetAutoFlushSize — get the auto flush size of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGetAutoFlushSize(<i>auto_flush_size_return</i>) T_INT4 *<i>auto_flush_size_return</i>;</pre>
Arguments	<i>auto_flush_size_return</i> — storage for auto flush size
Return Values	TRUE if the auto flush size was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetAutoFlushSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>auto_flush_size_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetAutoFlushSize
Description	TipcSrvGetAutoFlushSize gets the auto flush size of the connection to RTserver by calling TipcConnGetAutoFlushSize. The auto flush size is used to control how much outgoing data (in bytes) are buffered before being automatically flushed to RTserver. The default for the auto flush size is 8192 bytes. See TipcConnGetAutoFlushSize for more information on auto flush sizes. TipcSrvGetAutoFlushSize may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetAutoFlushSize gets the auto flush size of the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	Unlike other connections, the auto flush size T_IPC_NO_AUTO_FLUSH_SIZE cannot be used with the connection to RTserver to disable all automatic flushing of outgoing messages. If TipcSrvGetAutoFlushSize returns FALSE, it does not store a value in <i>auto_flush_size_return</i>.
See Also	TipcConnGetAutoFlushSize, TipcSrvSetAutoFlushSize

Examples This example creates a connection to RTserver, then gets and prints the auto flush size of the connection to RTserver:

```
T_INT4 auto_flush_size;

if (!TpcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}

if (!TpcSrvGetAutoFlushSize(&auto_flush_size)) {
    return; /* error */
}

TutOut("The connection to RTserver auto flush size is %d bytes.\n",
       auto_flush_size);
```

TipcSrvGetBlockMode

Name	TipcSrvGetBlockMode — get the block mode of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGetBlockMode(<i>block_mode_return</i>) T_BOOL *<i>block_mode_return</i>;</pre>
Arguments	<i>block_mode_return</i> — storage for block mode
Return Values	TRUE if block mode was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetBlockMode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>block_mode_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetBlockMode
Description	TipcSrvGetBlockMode gets the block mode of the connection to RTserver by calling TipcConnGetBlockMode. The block mode controls whether or not the RTclient blocks waiting for read and write operations to complete. See TipcConnGetBlockMode for more information on connection block modes. TipcSrvGetBlockMode may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetBlockMode does not use the warm connection, because the warm connection does not have a valid socket. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvGetBlockMode returns FALSE, it does not store a value in <i>block_mode_return</i> .
See Also	TipcConnGetBlockMode, TipcSrvSetBlockMode
Examples	This example gets the block mode of the connection to RTserver and prints it: <pre>T_BOOL block_mode; if (!TipcSrvGetBlockMode(&block_mode)) { return; /* error */ } TutOut("Connection to RTserver %s for read/write operations.\n", block_mode ? "blocks" : "does not block");</pre>

TpcSrvGetCurrentConnectedLcn

Name	TpcSrvGetCurrentConnectedLcn -- get the logical connection name of connected RTserver
Synopsis	<pre>T_BOOL TpcSrvGetCurrentConnectedLcn(lcn_return) T_STR lcn_return;</pre>
Arguments	<i>lcn_return</i> — storage for RTserver logical connection name
Return Values	TRUE if logical connection name was successfully retrieved from RTserver handle, FALSE otherwise.
Diagnostics	If TpcSrvGetConnStatus fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>lcn_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver does not exist
Description	TpcSrvGetCurrentConnectedLcn gets the logical connection name of the RTserver to which this RTclient is connected. If the function cannot find the logical connection name, it stores the string Unknown in <i>lcn_return</i>.
Caution	There is no way to know ahead of time how much storage is needed for <i>lcn_return</i>. The value of <i>lcn_return</i> cannot be NULL. Storage for <i>lcn_return</i> must be managed by the caller. If TpcSrvGetCurrentConnectedLcn returns FALSE, it does not store a value in <i>lcn_return</i>.
See Also	TpcSrvConnGetCurrentConnectedLcn
Examples	This example prints the name of the logical connection name of RTserver: <pre>T_CHAR lcn[80]; if (!TpcSrvGetCurrentConnectedLcn(lcn)) { return; /* error */ } TutOut("Current Logical Connection Name is <%s>\n", lcn);</pre>

TipcSrvGetConnStatus

Name	TipcSrvGetConnStatus — get the status of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGetConnStatus(<i>conn_status_return</i>) T_IPC_SRV_CONN_STATUS *<i>conn_status_return</i>;</pre>
Arguments	<i>conn_status_return</i> — storage for connection status
Return Values	TRUE if connection status was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcSrvGetConnStatus fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>conn_status_return</i> was null
Description	TipcSrvGetConnStatus gets the status of the connection to RTserver. A connection status of T_IPC_SRV_CONN_FULL indicates that the RTclient does have a connection to RTserver. A connection status of T_IPC_SRV_CONN_WARM indicates that the RTclient has a warm connection to RTserver. A connection status of T_IPC_SRV_CONN_NONE indicates that the RTclient does not have a connection to RTserver. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver. See TipcSrvDestroy for more information on destroying the connection to RTserver.
Caution	If TipcSrvGetConnStatus returns FALSE, it does not store a value in <i>conn_status_return</i>.
See Also	TipcSrvCreate, TipcSrvDestroy, TipcSrvIsRunning
Examples	This example uses TipcSrvGetConnStatus to ensure that the RTclient has some kind of connection to RTserver: <pre>T_IPC_SRV_CONN_STATUS conn_status; if (!TipcSrvGetConnStatus(&conn_status)) { return; /* error */ } if (conn_status == T_IPC_SRV_CONN_NONE) { TutOut("This process has no connection to RTserver.\n"); if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) { return; /* error */ } }</pre>

TipcSrvGetGmdMaxSize

Name	TipcSrvGetGmdMaxSize — get the GMD area maximum size of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGetGmdMaxSize(<i>gmd_max_size_return</i>) T_UINT4 *<i>gmd_max_size_return</i>;</pre>
Arguments	<i>gmd_max_size_return</i> — storage for connection to RTserver GMD area maximum size
Return Values	TRUE if the GMD area maximum size was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetGmdMaxSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>gmd_max_size_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetGmdMaxSize
Description	TipcSrvGetGmdMaxSize gets the GMD area maximum size (in bytes) of the connection to RTserver by calling TipcConnGetGmdMaxSize. A GMD area holds guaranteed message delivery information for both incoming and outgoing messages. There are two types of guaranteed message delivery: file-based GMD and memory-only GMD. See TipcConnGmdFileCreate for more information on GMD areas. For file-based GMD this limit is the maximum file size, and for memory-only GMD this limit is the maximum amount of memory allowed. The default GMD area maximum size is 0, which means that no GMD area maximum size limit checking is performed. The connection GMD area maximum size and the connection delivery timeout can be used to constrain the amount of system resources used for GMD by space and by time. If the connection GMD area maximum size is exceeded, then no further messages can be sent with guaranteed delivery (TipcSrvMsgSend returns FALSE and sets the global SmartSockets error number to T_ERR_VAL_TOO_LARGE) until some unacknowledged previously sent messages are acknowledged. TipcSrvGetGmdMaxSize may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetGmdMaxSize gets the GMD area maximum size of the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.

Caution If TipcSrvGetGmdMaxSize returns FALSE, it does not store a value in *gmd_max_size_return*.

See Also TipcConnGetGmdMaxSize, TipcSrvGmdFileCreate, TipcSrvSetGmdMaxSize

Examples This example prints the GMD area maximum size of the connection to RTserver:

```
T_INT4 gmd_max_size;

if (!TipcSrvGetGmdMaxSize(&gmd_max_size)) {
    return; /* error */
}

TutOut("Connection to RTserver GMD max size is %d.\n",
gmd_max_size);
```

TipcSrvGetGmdNumPending

Name	TipcSrvGetGmdNumPending — get the number of outgoing GMD messages still pending on the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGetGmdNumPending(<i>gmd_num_pending_return</i>) T_INT4 *<i>gmd_num_pending_return</i>;</pre>
Arguments	<i>gmd_num_pending_return</i> — storage for pending message count
Return Values	TRUE if the number of pending GMD messages was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetGmdNumPending fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>gmd_num_pending_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetGmdNumPending
Description	TipcSrvGetGmdNumPending gets the number of GMD messages published to RTserver that have not been acknowledged nor explicitly deleted from the GMD area. Messages are deleted from the GMD area when a GMD_ACK message is received and when a GMD_FAILURE message is processed. TipcSrvGetGmdNumPending may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetGmdNumPending gets the number of pending GMD messages from the warm connection. See TipcSrvCreate on creating connections and warm connections.
Caution	If TipcSrvGetGmdNumPending returns FALSE, it does not store a value in <i>gmd_num_pending_return</i> .
See Also	TipcConnGetGmdNumPending

Examples This example loops processing messages until there are no more outgoing GMD messages pending on the connection to RTserver:

```
T_INT4 num_pending;
do {
    if (!TipcSrvMainLoop(1.0)) {
        return; /* error */
    }
    if (!TipcSrvGetGmdNumPending(&num_pending)) {
        return; /* error */
    }
} while (num_pending > 0);
```

TipcSrvGetNode

Name	TipcSrvGetNode — get the node name of the connected RTserver
Synopsis	<pre>T_BOOL TipcSrvGetNode(<i>node_return</i>) T_STR *<i>node_return</i>;</pre>
Arguments	<i>node_return</i> — storage for RTserver node name
Return Values	TRUE if the node name was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetNode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>node_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetNode
Description	TipcSrvGetNode gets the node name of the RTserver this RTclient is connected to by calling TipcConnGetNode. The RTserver node name can be useful for diagnostic purposes. There is some overlap between TipcSrvGetNode and TipcMonServerGeneralPoll, but TipcSrvGetNode is easier to use if only the node name is needed instead of all RTserver general information. TipcSrvGetNode may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetNode does not use the warm connection, because the warm connection does not have a peer process with a node name. See TipcSrvCreate on creating connections and warm connections to RTserver.
Caution	If TipcSrvGetNode returns FALSE, it does not store a value in <i>node_return</i>. The value stored in <i>node_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcConnGetNode, TipcMonServerGeneralPoll, TipcSrvGetArch, TipcSrvGetPid, TipcSrvGetUniqueSubject, TipcSrvGetUser

Examples This example prints the node name of the connected RTserver:

```
T_STR node;

if (!TipcSrvGetNode(&node)) {
    return; /* error */
}
TutOut("RTserver node name is <%s>.\n", node);
```

TipcSrvGetNumQueued

Name	TipcSrvGetNumQueued — get number of queued messages from the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGetNumQueued(num_queued_return) T_INT4 *num_queued_return;</pre>
Arguments	<i>num_queued_return</i> — storage for number of queued messages
Return Values	TRUE if number of queued messages was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetNumQueued fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>num_queued_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetNumQueued
Description	TipcSrvGetNumQueued gets the number of queued messages from the connection to RTserver by calling TipcConnGetNumQueued. A connection has a priority queue of incoming messages. See TipcConnGetNumQueued for more information on connection message queues. TipcSrvGetNumQueued may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetNumQueued gets the number of queued messages from the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvGetNumQueued returns FALSE, it does not store a value in <i>num_queued_return</i> .
See Also	TipcConnGetNumQueued, TipcSrvMsgNext, TipcSrvMainLoop, TipcSrvRead

Examples This example prints the number of queued messages, then reads from the connection to RTserver until at least three messages are available:

```
T_INT4 num_queued;
if (!TipcSrvGetNumQueued(&num_queued)) {
    return; /* error */
}
TutOut("The connection to RTserver has %d queued messages.\n",
      num_queued);

while (num_queued < 3) {
    if (!TipcSrvRead(T_TIMEOUT_FOREVER)) {
        return; /* error */
    }
    if (!TipcSrvGetNumQueued(&num_queued)) {
        return; /* error */
    }
}
```

TipcSrvGetPid

Name	TipcSrvGetPid — get the process ID of the connected RTserver
Synopsis	<pre>T_BOOL TipcSrvGetPid(pid_return) T_INT4 *pid_return;</pre>
Arguments	<i>pid_return</i> — storage for RTserver ID
Return Values	TRUE if the process ID was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetPid fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>pid_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetPid
Description	TipcSrvGetPid gets the operating system process ID of the RTserver this RTclient is connected to by calling TipcConnGetPid. The RTserver ID can be useful for diagnostic purposes. There is some overlap between TipcSrvGetPid and TipcMonServerGeneralPoll, but TipcSrvGetPid is easier to use if only the process ID is needed instead of all RTserver general information. TipcSrvGetPid may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetPid does not use the warm connection, because the warm connection does not have a peer process with a process ID. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvGetPid returns FALSE, it does not store a value in <i>pid_return</i> .
See Also	TipcConnGetPid, TipcMonServerGeneralPoll, TipcSrvGetArch, TipcSrvGetNode, TipcSrvGetUniqueSubject, TipcSrvGetUser
Examples	This example prints the process ID of the connected RTserver: <pre>T_INT4 pid; if (!TipcSrvGetPid(&pid)) { return; /* error */ } TutOut("RTserver ID is <%d>.\n", pid);</pre>

TipcSrvGetSocket

Name	TipcSrvGetSocket — get the socket of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGetSocket(socket_return) T_INT4 *socket_return;</pre>
Arguments	<i>socket_return</i> — storage for connection socket
Return Values	TRUE if socket was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetSocket fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>socket_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetSocket
Description	TipcSrvGetSocket gets the socket file descriptor of the connection to RTserver by calling TipcConnGetSocket. A socket is an operating system device which provides a communication link to another process. All data sent to and received from the connection is transmitted on this socket. TipcSrvGetSocket may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetSocket does not use the warm connection, because the warm connection does not have a valid socket. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvGetSocket returns FALSE, it does not store a value in <i>socket_return</i> .
See Also	TipcConnGetSocket, TipcSrvSetSocket, TipcSrvGetXtSource
Examples	This example creates a connection to RTserver and prints the socket of the connection: <pre>T_INT4 socket; if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) { return; /* error */ } if (!TipcSrvGetSocket(&socket)) { return; /* error */ } TutOut("Socket of connection to RTserver is <%d>.\n", socket);</pre>

TipcSrvGetTimeout

Name	TipcSrvGetTimeout — get a timeout property from the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGetTimeout(<i>timeout</i>, <i>value_return</i>) T_IPC_TIMEOUT <i>timeout</i>; T_REAL8 *<i>value_return</i>;</pre>
Arguments	<i>timeout</i> — which timeout property to get <i>value_return</i> — storage for timeout value
Return Values	TRUE if the timeout property was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetTimeout fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>timeout</i> was not a valid T_IPC_TIMEOUT • T_ERR_NULL_PTR — <i>value_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetTimeout
Description	TipcSrvGetTimeout gets a timeout property from the connection to RTserver by calling TipcConnGetTimeout. The timeout properties control how often (in seconds) the connection to RTserver is checked for possible network failures. See TipcConnGetTimeout for more information on connection timeouts. The timeout properties of the connection to RTserver are also available in the options Server_Read_Timeout, Server_Write_Timeout, Server_Keep_Alive_Timeout, and Server_Delivery_Timeout. For the connection to RTserver, the default read timeout is 30.0 seconds, the default write timeout is 30.0 seconds, the default keep alive timeout is 5.0 seconds, and the default delivery timeout is 30.0 seconds. TipcSrvGetTimeout may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetTimeout gets a timeout property from the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvGetTimeout returns FALSE, it does not store a value in <i>value_return</i> .
See Also	TipcConnGetTimeout, TipcSrvSetTimeout

Examples This example gets the read timeout from the connection to RTserver with TipcSrvGetTimeout and prints it:

```
T_REAL8 timeout;

if (!TipcSrvGetTimeout(T_IPC_TIMEOUT_READ, &timeout)) {
    return; /* error */
}

TutOut("The connection to RTserver's read timeout is %s
seconds.\n",
        TutRealToStr(timeout));
```

TpcSrvGetUniqueSubject

Name	TpcSrvGetUniqueSubject — get the unique subject of the connected RTserver
Synopsis	<pre>T_BOOL TpcSrvGetUniqueSubject(unique_subject_return) T_STR *unique_subject_return;</pre>
Arguments	<i>unique_subject_return</i> — storage for RTserver unique subject
Return Values	TRUE if the unique subject was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TpcSrvGetUniqueSubject fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>unique_subject_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TpcConnGetUniqueSubject
Description	TpcSrvGetUniqueSubject gets the unique subject of the RTserver this RTclient is connected to by calling TpcConnGetUniqueSubject. The RTserver unique subject can be useful for diagnostic purposes. The unique subject, which is used to uniquely identify a process, is stored in the option Unique_Subject. There is some overlap between TpcSrvGetUniqueSubject and TpcMonServerGeneralPoll, but TpcSrvGetUniqueSubject is easier to use if only the unique subject is needed instead of all RTserver general information. TpcSrvGetUniqueSubject may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TpcSrvGetUniqueSubject does not use the warm connection, because the warm connection does not have a peer process with a unique subject. See TpcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	If TpcSrvGetUniqueSubject returns FALSE, it does not store a value in <i>unique_subject_return</i>. The value stored in <i>unique_subject_return</i> points directly into an internal data structure and should not be modified.
See Also	TpcConnGetUniqueSubject, TpcMonServerGeneralPoll, TpcSrvGetArch, TpcSrvGetNode, TpcSrvGetPid, TpcSrvGetUser

Examples This example prints the unique subject of the connected RTserver:

```
T_STR unique_subject;

if (!TipcSrvGetUniqueSubject(&unique_subject)) {
    return; /* error */
}

TutOut("RTserver unique subject is <%s>.\n",
        unique_subject);
```

TipcSrvGetUser

Name	TipcSrvGetUser — get the user name of the connected RTserver
Synopsis	<pre>T_BOOL TipcSrvGetUser(<i>user_return</i>) T_STR *<i>user_return</i>;</pre>
Arguments	<i>user_return</i> — storage for RTserver user name
Return Values	TRUE if the user name was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetUser fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>user_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetUser
Description	TipcSrvGetUser gets the user name of the RTserver this RTclient is connected to by calling TipcConnGetUser. The RTserver username can be useful for diagnostic purposes. There is some overlap between TipcSrvGetUser and TipcMonServerGeneralPoll, but TipcSrvGetUser is easier to use if only the user name is needed instead of all RTserver general information. TipcSrvGetUser may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetUser does not use the warm connection, because the warm connection does not have a peer process with a user name. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvGetUser returns FALSE, it does not store a value in <i>user_return</i>. The value stored in <i>user_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcConnGetUser, TipcMonServerGeneralPoll, TipcSrvGetArch, TipcSrvGetNode, TipcSrvGetPid, TipcSrvGetUniqueSubject

Examples This example prints the user name of the connected RTserver:

```
T_STR user;
if (!TipcSrvGetUser(&user)) {
    return; /* error */
}
TutOut("RTserver user name is <%s>.\n", user);
```

TipcSrvGetXtSource

Name	TipcSrvGetXtSource — get source suitable for XtAppAddInput from the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGetXtSource(<i>source_return</i>) T_INT4 *<i>source_return</i>;</pre>
Arguments	<i>source_return</i> — storage for source
Return Values	TRUE if source was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGetXtSource fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>source_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetXtSource
Description	TipcSrvGetXtSource gets an XtAppAddInput-compatible source from the connection to RTserver by calling TipcConnGetXtSource. This source allows SmartSockets IPC to be mixed with Motif or any other Xt-based widget set. See TipcConnGetXtSource for more information on Xt-compatible sources. TipcSrvGetXtSource may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGetXtSource does not use the warm connection, because the warm connection does not have a valid source. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvGetXtSource returns FALSE, it does not store a value in <i>source_return</i> .
See Also	TipcConnGetXtSource, TipcSrvGetSocket, TipcSrvSetSocket

Examples This example creates a connection to RTserver, then gets and prints the Xt-compatible source of the connection:

```
T_INT4 source;

if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}
if (!TipcSrvGetXtSource(&source)) {
    return; /* error */
}
TutOut("Source of connection to RTserver is <%d>.\n", source);
```

TipcSrvGmdFileCreate

Name	TipcSrvGmdFileCreate — create guaranteed message delivery area on the connection to RTserver
Synopsis	<code>T_BOOL TipcSrvGmdFileCreate()</code>
Arguments	None
Return Values	TRUE if the GMD area was successfully created for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGmdFileCreate fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_SRV_NOT_CONNECTED</code> — a connection to RTserver could not be created • any error number from <code>TipcConnGmdFileCreate</code>
Description	TipcSrvGmdFileCreate creates a GMD area on the connection to RTserver by calling <code>TipcConnGmdFileCreate</code>. A GMD area holds guaranteed message delivery information for both incoming and outgoing messages. There are two types of guaranteed message delivery: file-based GMD and memory-only GMD. See <code>TipcConnGmdFileCreate</code> for more information on GMD areas. The full pathnames of the GMD files for the connection to RTserver are simpler than those for other connections. The full pathnames are built from the value from <code>TipcGetGmdDir</code> plus <code>rtserver</code>. For example, an RTclient process with the unique subject <code>unique_subject</code> would use the full pathname: UNIX <pre>/tmp/rtworks/gmd/unique_subject/rtserver.hsn</pre> OpenVMS <pre>dka300:[ssuser.gmd.unique_subject]rtserver.hsn</pre> Windows <pre>c:\temp\rtworks\gmd\unique_subject\rtserver.hsn</pre> Because the connection to RTserver has a fixed purpose, it can use shorter file names than other generic connections.

TipcSrvGmdFileCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvGmdFileCreate creates the GMD area for the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.

Caution TipcSrvCreate calls TipcSrvGmdResend to automatically resend old GMD messages, and TipcSrvGmdResend calls TipcSrvGmdFileCreate if old GMD files exist.

The most common cause of TipcSrvGmdFileCreate switching to memory-only GMD is not setting the option `Unique_Subject` to a value other than the default `_Node_Pid`. Each process that uses file-based GMD must properly configure `Unique_Subject`.

See Also TipcConnGmdFileCreate, TipcSrvGmdFileDelete

Examples This example calls TipcSrvGmdFileCreate to explicitly create the GMD area for the connection to RTserver, and then sends a `NUMERIC_DATA` message with guaranteed delivery to the test subject through RTserver:

```
T_IPC_MT mt;
T_IPC_MSG msg;

/* have to set the Unique_Subject option to use file-based GMD */
TutCommandParseStr("setopt unique_subject /example/sender");

/* Explicitly create GMD area to allow full control over error handling. This is slightly tricky since
TipcSrvGmdFileCreate may call TipcSrvCreate which calls TipcSrvGmdResend which calls
TipcSrvGmdFileCreate, so the first call can fail with error number T_ERR_ALREADY_EXISTS. */
if (!TipcSrvGmdFileCreate()
    && TutErrNumGet() != T_ERR_ALREADY_EXISTS) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (!TipcSrvMsgWrite("/system/test", mt, FALSE,
                    T_IPC_PROP_DELIVERY_MODE, T_IPC_DELIVERY_ALL,
                    T_IPC_FT_STR, "test_value",
                    T_IPC_FT_REAL8, 10.0,
                    NULL)) {
    return; /* error */
}
```

TipcSrvGmdFileDelete

Name	TipcSrvGmdFileDelete — delete guaranteed message delivery files for the connection to RTserver
Synopsis	<code>T_BOOL TipcSrvGmdFileDelete()</code>
Arguments	None
Return Values	TRUE if the connection GMD files were successfully deleted from disk, FALSE otherwise.
Diagnostics	If TipcSrvGmdFileDelete fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_IN_USE</code> — the connection GMD files are already open by this process and cannot be removed • <code>T_ERR_C</code> — a C error has occurred. Call the function <code>TutErrNumGetC</code> to obtain the related C-specific error value. • any error number from <code>TipcGetGmdDir</code>
Description	TipcSrvGmdFileDelete deletes from disk guaranteed message delivery (GMD) files for the connection to RTserver. Because TipcSrvCreate calls TipcSrvGmdResend (which opens any existing GMD files), TipcSrvGmdFileDelete does not call TipcConnGmdFileDelete but instead uses the same algorithm as TipcConnGmdFileDelete. TipcSrvGmdFileDelete is useful for processes that wish to completely delete any obsolete GMD information. See TipcConnGmdFileDelete for more information on deleting GMD files.
Caution	Old GMD files must be removed before a warm or full connection to RTserver is created. TipcSrvGmdFileDelete does not delete the GMD files if they are already open by this process and in use for the connection to RTserver.
See Also	TipcConnGmdFileDelete, TipcSrvGmdFileCreate

Examples This example calls TipcSrvGmdFileDelete to remove any existing GMD files for the connection to RTserver, creates a connection to RTserver, and then sends a NUMERIC_DATA message with guaranteed delivery to the test subject using RTserver:

```
T_IPC_MT mt;
T_IPC_MSG msg;

/* have to set the Unique_Subject option to use file-based GMD */
TutCommandParseStr("setopt unique_subject /example/sender");

/* Clear out old GMD files before connecting to RTserver. No special action is needed if you wish to
instead resend any previously sent messages with GMD (TipcSrvCreate automatically calls
TipcSrvGmdResend). */
if (!TipcSrvGmdFileDelete()) {
    return; /* error */
}

if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (!TipcSrvMsgWrite("/system/test", mt, FALSE,
                    T_IPC_PROP_DELIVERY_MODE, T_IPC_DELIVERY_ALL,
                    T_IPC_FT_STR, "test_value",
                    T_IPC_FT_REAL8, 10.0,
                    NULL)) {
    return; /* error */
}
```

TipcSrvGmdMsgDelete

Name	TipcSrvGmdMsgDelete — delete a message from GMD area after a GMD failure on the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGmdMsgDelete(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to delete
Return Values	TRUE if the guaranteed message was successfully deleted from the GMD area of the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvGmdMsgDelete fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — the message was not previously sent with GMD • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGmdMsgDelete
Description	TipcSrvGmdMsgDelete deletes a message after a GMD failure on the connection to RTserver by calling TipcConnGmdMsgDelete. TipcSrvGmdMsgDelete is intended to be used from a GMD_FAILURE connection process callback to terminate GMD for the message. See TipcConnGmdMsgDelete for more information on deleting messages when GMD failures occur. When a GMD_FAILURE message is processed by a sender RTclient, one of two scenarios is possible: • the sender RTclient does not want to resend the message, and thus uses TipcSrvGmdMsgDelete and TipcSrvGmdMsgServerDelete to terminate GMD for the message • the sender RTclient does want to resend the message, and thus takes some kind of user-defined action to correct the problem (possibly using TipcSrvGmdMsgStatus to poll RTserver for GMD status) and then resends the message

The standard callback `TipcCbConnProcessGmdFailure` uses `TipcConnGmdMsgDelete` to delete failed messages. Use `TipcSrvGmdMsgDelete` instead of `TipcConnGmdMsgDelete` in user-defined callbacks for processing `GMD_FAILURE` messages on the connection to RTserver. See `TipcCbConnProcessGmdFailure` and `TipcCbSrvProcessGmdFailure` for more information on processing `GMD_FAILURE` messages.

`TipcSrvGmdMsgDelete` may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, `TipcSrvGmdMsgDelete` deletes the message from the GMD area for the warm connection. See `TipcSrvCreate` on automatic creation of connections and warm connections to RTserver.

Caution Use `TipcSrvGmdMsgDelete` only from a `GMD_FAILURE` process callback.

See Also `TipcCbConnProcessGmdFailure`, `TipcCbSrvProcessGmdFailure`, `TipcConnGmdMsgDelete`, `TipcSrvGmdMsgStatus`

Examples This example creates a connection process callback for `GMD_FAILURE` messages which silently deletes the message that failed (this callback is essentially a silent version of `TipcCbConnProcessGmdFailure`). The existing callback `TipcCbConnProcessGmdFailure` is destroyed so as not to interfere with the new callback:

```
void T_ENTRY my_gmd_failure_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_IPC_MSG failed_msg;

    /* go to the first field of the GMD_FAILURE message */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* read the first field of the GMD_FAILURE message, which is */
    /* the message where GMD failed */
    if (!TipcMsgNextMsg(data->msg, &failed_msg))
        return; /* error */
    }

    /* terminate the sender's knowledge about the message for GMD */
    if (!TipcSrvGmdMsgDelete(failed_msg)) {
        return; /* error */
    }
} /* my_gmd_failure_cb */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;
T_CB cb;

mt = TipcMtLookupByNum(T_MT_GMD_FAILURE);
if (mt == NULL) {
    return; /* error */
}

/* create connection process callback for GMD_FAILURE messages */
if (TipcSrvProcessCbCreate(mt, my_gmd_failure_cb, NULL)
    == NULL) {
    return; /* error */
}

/* destroy standard callback for GMD_FAILURE messages */
cb = TipcSrvProcessCbLookup(mt, TipcCbConnProcessGmdFailure,
NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}

```

TipcSrvGmdMsgServerDelete

Name	TipcSrvGmdMsgServerDelete — delete a message in RTserver after a GMD failure on the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvGmdMsgServerDelete(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message for RTserver to delete
Return Values	TRUE if deletion of the message from RTserver was successfully initiated, FALSE otherwise.
Diagnostics	If TipcSrvGmdMsgServerDelete fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — the message was not previously sent with GMD • any error number from TipcSrvMsgWrite
Description	TipcSrvGmdMsgServerDelete informs RTserver to delete a message after a GMD failure on the connection to RTserver. TipcSrvGmdMsgServerDelete sends a GMD_DELETE message to RTserver. When RTserver routes a message for GMD, it keeps the message in memory so that it can easily and quickly resend the message to any receiving RTclient that disconnects and reconnects. TipcSrvGmdMsgServerDelete informs RTserver to terminate GMD for the message, which allows RTserver to reclaim the memory for the message. TipcSrvGmdMsgServerDelete is intended to be used from a GMD_FAILURE connection process callback to terminate GMD for the message. See TipcSrvGmdMsgDelete for more information on deleting messages when GMD failures occur. When a GMD_FAILURE message is processed by a sender RTclient, one of two scenarios is possible: • the sender RTclient does not want to resend the message, and thus uses TipcSrvGmdMsgDelete and TipcSrvGmdMsgServerDelete to terminate GMD for the message • the sender RTclient does want to resend the message, and thus takes some kind of user-defined action to correct the problem (possibly using TipcSrvGmdMsgStatus to poll RTserver for GMD status) and then resends the message

The standard callback `TipcCbSrvProcessGmdFailure` uses `TipcSrvGmdMsgServerDelete` to delete failed messages. See `TipcCbConnProcessGmdFailure` and `TipcCbSrvProcessGmdFailure` for more information on processing `GMD_FAILURE` messages.

Caution Use `TipcSrvGmdMsgServerDelete` only from a `GMD_FAILURE` process callback. `TipcSrvGmdMsgServerDelete` sends a `GMD_DELETE` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

See Also `TipcCbConnProcessGmdFailure`, `TipcCbSrvProcessGmdFailure`, `TipcConnGmdMsgDelete`, `TipcSrvGmdMsgStatus`

Examples This example creates a connection process callback for `GMD_FAILURE` messages which takes corrective actions to recover from the GMD failure. If recovery succeeds, then the message is resent, otherwise the message is deleted in the sender and in `RTserver`. The existing callbacks `TipcCbConnProcessGmdFailure` and `TipcCbSrvProcessGmdFailure` are destroyed so as to not interfere with the new callback:

```
void T_ENTRY my_gmd_failure_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_IPC_MSG failed_msg;
    T_STR failed_proc_name;
    T_INT4 err_num;
    T_REAL8 send_time;

    /* go to the first field of the GMD_FAILURE message */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* read the fields of the GMD_FAILURE message */
    if (!TipcMsgNextMsg(data->msg, &failed_msg)
        || !TipcMsgNextStr(data->msg, &failed_proc_name)
        || !TipcMsgNextInt4(data->msg, &err_num)
        || !TipcMsgNextReal8(data->msg, &send_time)) {
        return; /* error */
    }

    /* at this point, a sender would typically take some */
    /* application-specific corrective action before resending */
    /* the message */
    if (take_corrective_actions(failed_proc_name, err_num,
        send_time)) {
```



```

        /* resend the message with GMD */
        if (!TipcSrvMsgSend(failed_msg, TRUE)) {
            return; /* error */
        }
    }
    else {
        /* delete the message in the sender (this process) */
        if (!TipcSrvGmdMsgDelete(failed_msg)) {
            return; /* error */
        }

        /* delete the message in RTserver */
        if (!TipcSrvGmdMsgServerDelete(failed_msg)) {
            return; /* error */
        }
    }
} /* my_gmd_failure_cb */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;
T_CB cb;

mt = TipcMtLookupByNum(T_MT_GMD_FAILURE);
if (mt == NULL) {
    return; /* error */
}

/* create connection process callback for GMD_FAILURE messages */
if (TipcSrvProcessCbCreate(mt, my_gmd_failure_cb, NULL) == NULL) {
    return; /* error */
}

/* destroy standard callbacks for GMD_FAILURE messages */

cb = TipcSrvProcessCbLookup(mt, TipcCbConnProcessGmdFailure,
NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}

cb = TipcSrvProcessCbLookup(mt, TipcCbSrvProcessGmdFailure, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}
}

```

TipcSrvGmdMsgStatus

Name	TipcSrvGmdMsgStatus — poll RTserver for GMD status of a message
Synopsis	<pre>T_BOOL TipcSrvGmdMsgStatus(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to poll RTserver about
Return Values	TRUE if the poll for GMD status was successfully initiated, FALSE otherwise.
Diagnostics	If TipcSrvGmdMsgStatus fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — the message was not previously sent with GMD • any error number from TipcSrvMsgWrite
Description	TipcSrvGmdMsgStatus polls for the GMD status of <i>msg</i> by sending a GMD_STATUS_CALL message to RTserver. RTserver responds by sending back a GMD_STATUS_RESULT message. The response should come back very quickly. The polling RTclient can use functions such as TipcSrvMsgSearchType to get the response. When RTserver routes a message for GMD, it keeps track of which RTclients should receive the message and the status of GMD to each of these RTclients. TipcSrvGmdMsgStatus is intended to be used from a GMD_FAILURE connection process callback to query GMD status for the message. The GMD_STATUS_RESULT message contains four fields: • an INT4 integer field containing the sequence number property of the message • a STR_ARRAY string array field containing the names of the RTclients that have acknowledged successful delivery of the message • a STR_ARRAY string array field containing the names of the RTclients where GMD has failed for the message • a STR_ARRAY string array field containing the names of the RTclients where delivery is still pending for the message

If RTserver does not have an in-memory GMD record of *msg*, RTserver silently ignores the GMD_STATUS_CALL message and no GMD_STATUS_RESULT message is returned. This indicates that one of these has happened:

- RTserver has fully delivered *msg*
- TipcSrvGmdMsgServerDelete was used to terminate GMD for the message
- RTserver never routed *msg*

When a GMD_FAILURE message is processed by a sender RTclient, one of two scenarios is possible:

- the sender RTclient does not want to resend the message, and thus uses TipcSrvGmdMsgDelete and TipcSrvGmdMsgServerDelete to terminate GMD for the message
- the sender RTclient does want to resend the message, and thus takes some kind of user-defined action to correct the problem (possibly using TipcSrvGmdMsgStatus to poll RTserver for GMD status) and then resends the message

See TipcCbConnProcessGmdFailure and TipcCbSrvProcessGmdFailure for more information on processing GMD_FAILURE messages.

Caution TipcSrvGmdMsgStatus is usually only used from a GMD_FAILURE process callback, but it can be called at any time by advanced applications which wish to poll for GMD status.

TipcSrvGmdMsgStatus sends a GMD_STATUS_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

See Also TipcCbConnProcessGmdFailure, TipcCbSrvProcessGmdFailure, TipcConnGmdMsgDelete, TipcSrvGmdMsgServerDelete

Examples This example creates a connection process callback for GMD_FAILURE messages which uses TipcSrvGmdMsgStatus to poll RTserver for GMD status. (This example callback simply prints the GMD status information. A real callback would take some corrective actions to recover from the GMD failure.)

```
void T_ENTRY my_gmd_failure_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_IPC_MSG failed_msg;
    T_IPC_MT mt;
    T_IPC_MSG result_msg;
    T_INT4 seq_num;
    T_STR *success_client_names;
    T_INT4 num_success_client_names;
    T_STR *failure_client_names;
    T_INT4 num_failure_client_names;
    T_STR *pending_client_names;
    T_INT4 num_pending_client_names;
    T_INT4 counter; /* loop counter */

    /* go to the first field of the GMD_FAILURE message */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* read the first field of the GMD_FAILURE message, which is */
    /* the message where GMD failed */
    if (!TipcMsgNextMsg(data->msg, &failed_msg)) {
        return; /* error */
    }

    /* poll RTserver for GMD status information */
    if (!TipcSrvGmdMsgStatus(failed_msg)) {
        return; /* error */
    }

    mt = TipcMtLookupByNum(T_MT_GMD_STATUS_RESULT);
    if (mt == NULL) {
        return; /* error */
    }

    /* wait a short time for the GMD status result */
    result_msg = TipcSrvMsgSearchType(10.0, mt);
    if (result_msg == NULL) {
        return; /* error */
    }

    /* must set current field first */
    if (!TipcMsgSetCurrent(result_msg, 0)) {
        return; /* error */
    }
}
```

```

/* get the fields from the message */
if (!TipcMsgRead(result_msg,
                 T_IPC_FT_INT4, &seq_num,
                 T_IPC_FT_STR_ARRAY, &success_client_names,
                 &num_success_client_names,
                 T_IPC_FT_STR_ARRAY, &failure_client_names,
                 &num_failure_client_names,
                 T_IPC_FT_STR_ARRAY, &pending_client_names,
                 &num_pending_client_names,
                 NULL)) {
    return; /* error */
}

TutOut("Got GMD_STATUS poll result.\n");
TutOut("seq_num = %d\n", seq_num);
TutOut("success_client_names\n");
for (counter = 0; counter < num_success_client_names; counter++)
{
    TutOut(" %s\n", success_client_names[counter]);
}
TutOut("failure_client_names\n");
for (counter = 0; counter < num_failure_client_names; counter++)
{
    TutOut(" %s\n", failure_client_names[counter]);
}
TutOut("pending_client_names\n");
for (counter = 0; counter < num_pending_client_names; counter++)
{
    TutOut(" %s\n", pending_client_names[counter]);
}

if (!TipcMsgDestroy(result_msg)) {
    return; /* error */
}
} /* my_gmd_failure_cb */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;
T_CB cb;

mt = TipcMtLookupByNum(T_MT_GMD_FAILURE);
if (mt == NULL) {
    return; /* error */
}

/* create connection process callback for GMD_FAILURE messages */
cb = TipcSrvProcessCbCreate(mt, my_gmd_failure_cb, NULL);
if (cb == NULL) {
    return; /* error */
}

```

```
/* raise priority of callback so it is called before */  
/* TpcCbSrvProcessGmdFailure, which uses */  
/* TpcSrvGmdMsgServerDelete to tell RTserver to forget */  
/* the message */  
if (!TutCbSetPriority(cb, 2)) {  
    return; /* error */  
}
```

TipcSrvIsRunning

Name	TipcSrvIsRunning — check if the RTclient can create a connection to RTserver
Synopsis	<code>T_BOOL TipcSrvIsRunning()</code>
Arguments	None
Return Values	TRUE if the RTclient can create a connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvIsRunning fails, it returns FALSE and sets the global SmartSockets error number to: • <code>T_ERR_SRV_NOT_CONNECTED</code> — a connection to RTserver could not be created
Description	TipcSrvIsRunning checks if the RTclient can create a connection to RTserver. If the RTclient already has a full connection to RTserver, then TipcSrvIsRunning returns TRUE immediately. Otherwise TipcSrvIsRunning attempts to create a connection in a fashion similar to that of TipcSrvCreate, except that TipcSrvIsRunning never attempts to start RTserver. If TipcSrvIsRunning can create a connection, it destroys the connection and return TRUE.
Caution	TipcSrvIsRunning actually tries to create a connection to RTserver.
See Also	TipcSrvCreate, TipcSrvDestroy
Examples	This example checks if RTserver is already running with TipcSrvIsRunning: <pre>if (TipcSrvIsRunning()) { TutOut("RTserver is running.\n"); } else { TutOut("RTserver is not running.\n"); }</pre>

TipcSrvKeepAlive

Name	TipcSrvKeepAlive — check the health of the connection to RTserver.
Synopsis	T_BOOL TipcSrvKeepAlive()
Arguments	None
Return Values	TRUE if RTserver responded in time, FALSE otherwise.
Diagnostics	If TipcSrvKeepAlive fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnKeepAlive
Description	TipcSrvKeepAlive checks if the connection to RTserver is still alive by calling TipcConnKeepAlive. This check is called a keep alive. See TipcConnKeepAlive for more information on keep alives. TipcSrvKeepAlive may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvKeepAlive immediately returns FALSE. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnKeepAlive
Examples	This example creates a connection to RTserver, then performs a keep alive every 10 seconds to check the health of the connection to RTserver: <pre>if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) { return; /* error */ } for (;;) { TutSleep(10.0); if (!TipcSrvKeepAlive()) { return; /* error */ } }</pre>

TipcSrvLock

Name	TipcSrvLock — acquire exclusive access to the connection to RTserver
Synopsis	<code>T_BOOL TipcSrvLock()</code>
Arguments	None
Return Values	TRUE if the connection to RTserver was successfully locked, FALSE otherwise.
Diagnostics	If TipcSrvLock fails, it returns FALSE and sets the global SmartSockets error number to one of: any error number from TutRwMutexWriteLock
Description	TipcSrvLock locks all of the internal synchronization objects within the connection to RTserver. This insures that the calling thread in a multi-threaded process has exclusive access to the connection to RTserver. TipcSrvLock may be called by a thread recursively.
Caution	Every call to TipcSrvLock must be balanced by a call to TipcSrvUnlock.
See Also	TipcConnLock, TipcSrvUnlock
Examples	This example locks and unlocks the connection to RTserver: <pre> if (!TipcSrvLock()) { return; /* error */ } /* At this point, several TipcSrv* functions could be called and there would be no risk of any other thread accessing the connection. The connection must be unlocked with TipcSrvUnlock when exclusive access is no longer needed. */ if (!TipcSrvUnlock()) { return; /* error */ } </pre>

TipcSrvLogAddMt

Name	TipcSrvLogAddMt — add a message type to a message file logging type
Synopsis	<pre>T_BOOL TipcSrvLogAddMt(log_type, mt) T_IPC_SRV_LOG_TYPE log_type; T_IPC_MT mt;</pre>
Arguments	<i>log_type</i> — message file logging type <i>mt</i> — message type to add to list
Return Values	TRUE if the message type was successfully added, FALSE otherwise.
Diagnostics	If TipcSrvLogAddMt fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> was null • T_ERR_VAL_INVALID — <i>log_type</i> was not a valid T_IPC_SRV_LOG_TYPE • T_ERR_SRV_ALREADY_CONNECTED — the RTclient already has a connection to RTserver • T_ERR_ALREADY_EXISTS — <i>mt</i> already exists in the list • any error number from TutPtrAryAppend
Description	TipcSrvLogAddMt adds a message type to a message file logging type on the connection to RTserver. Each logging type has two options that can be used to log incoming and outgoing messages in that logging type. When the option is set (the option value should be the name of a file), then read callbacks (for incoming) or write callbacks (for outgoing) are created for all the logging type’s message types in the connection to RTserver. More than one option can use the same file. The logging type T_IPC_SRV_LOG_DATA, which is for variable value message types such as TIME and NUMERIC_DATA, is controlled by the options Log_In_Data and Log_Out_Data. The logging type T_IPC_SRV_LOG_STATUS, which is for RThci messages types such as ALERT and POPUP, is controlled by the options Log_In_Status and Log_Out_Status. The logging type T_IPC_SRV_LOG_INTERNAL, which is for all other message types such as SUBJECT_SET_SUBSCRIBE and SERVER_STOP_CALL, is controlled by the options Log_In_Internal and Log_Out_Internal. All of the standard SmartSockets message types are added automatically to one logging type. Message types can be added to multiple logging types.

Caution TipcSrvLogAddMt can only be called before the RTclient has created a connection to RTserver.

See Also TipcSrvLogRemoveMt; see the *TIBCO SmartSockets Utilities* for information on TutPtrArytAppend.

Examples This example creates a message type and adds it to the data logging type with TipcSrvLogAddMt:

```
#define USER_MT_COORD_DATA 100

mt = TipcMtCreate("coord_data", USER_MT_COORD_DATA,
                 "int4 /*x*/ int4 /*y*/ int4 /*z*/");
if (mt == NULL) {
    return; /* error */
}

if (!TipcSrvLogAddMt(T_IPC_SRV_LOG_DATA, mt)) {
    return; /* error */
}

TutCommandParseStr("setopt log_in_data data.msg");
/*
Data messages that arrive from RTserver (include those of type
USER_MT_COORD_DATA) will now be logged to the file data.msg.
*/
```

TipcSrvLogRemoveMt

Name	TipcSrvLogRemoveMt — remove a message type from a message file logging type
Synopsis	<pre>T_BOOL TipcSrvLogRemoveMt(log_type, mt) T_IPC_SRV_LOG_TYPE log_type; T_IPC_MT mt;</pre>
Arguments	<i>log_type</i> — message file logging type <i>mt</i> — message type to remove from list
Return Values	TRUE if the message type was successfully removed, FALSE otherwise.
Diagnostics	If TipcSrvLogRemoveMt fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> was null • T_ERR_VAL_INVALID — <i>log_type</i> was not a valid T_IPC_SRV_LOG_TYPE • T_ERR_SRV_ALREADY_CONNECTED — the RTclient already has a connection to RTserver • any error number from TutPtrAryRemove
Description	TipcSrvLogRemoveMt removes a message type from a message file logging type on the connection to RTserver. Each logging type has two options that can be used to log incoming and outgoing messages in that logging type. See TipcSrvLogAddMt for more information on logging types.
Caution	TipcSrvLogRemoveMt can only be called before the RTclient has created a connection to RTserver.
See Also	TipcSrvLogAddMt; see the <i>TIBCO SmartSockets Utilities</i> for information on TutPtrAryRemove.
Examples	This example removes the TIME message type from the data logging type with TipcSrvLogRemoveMt: <pre>mt = TipcMtLookupByNum(T_MT_TIME); if (mt == NULL) { return; /* error */ } if (!TipcSrvLogRemoveMt(T_IPC_SRV_LOG_DATA, mt)) { return; /* error */ }</pre>

TipcSrvMainLoop

Name	TipcSrvMainLoop — read and process messages on the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvMainLoop(<i>timeout</i>) T_REAL8 <i>timeout</i>;</pre>
Arguments	<i>timeout</i> — maximum number of seconds to wait for message to arrive
Return Values	TRUE if all pending messages were successfully processed on the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvMainLoop fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMainLoop
Description	TipcSrvMainLoop reads and processes messages on the connection to RTserver by calling TipcConnMainLoop. TipcSrvMainLoop is a convenience function that can be used in many situations instead of TipcSrvMsgNext, TipcSrvMsgProcess, and TipcMsgDestroy. See TipcConnMainLoop for more information on reading and processing messages. If an error occurs, TipcSrvMainLoop by default restarts RTserver and return FALSE. To use TipcSrvMainLoop with a timeout of T_TIMEOUT_FOREVER, wrap an extra forever loop, either for (; ;) or while (1), around the call to TipcSrvMainLoop. TipcSrvMainLoop may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvMainLoop processes messages on the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnMainLoop, TipcMsgDestroy, TipcSrvMsgNext, TipcSrvMsgProcess

Examples This example creates a connection to RTserver, then reads and processes messages forever with TpcSrvMainLoop (note the use of TpcSrvMainLoop in a loop):

```
if (!TpcSrvCreate(T_IPC_SRV_CONN_FULL)) {  
    return; /* error */  
}  
  
for (;;) {  
    if (!TpcSrvMainLoop(T_TIMEOUT_FOREVER)) {  
        return; /* error */  
    }  
}
```

TipcSrvMsgInsert

Name	TipcSrvMsgInsert — insert message into queue of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvMsgInsert(msg, pos) T_IPC_MSG msg; T_INT4 pos;</pre>
Arguments	<i>msg</i> — message to insert <i>pos</i> — position to insert message
Return Values	TRUE if the message was successfully inserted into the queue in the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvMsgInsert fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — <i>pos</i> was negative or larger than the number of messages in the queue • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgInsert
Description	TipcSrvMsgInsert inserts a message into the message queue of the connection to RTserver by calling TipcConnMsgInsert. A connection has a priority queue of incoming messages. See TipcConnMsgInsert for more information on inserting a message into a connection message queue. TipcSrvMsgInsert may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvMsgInsert inserts the message into the message queue of the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	Do not call TipcMsgDestroy to destroy the message passed to TipcSrvMsgInsert, as TipcSrvMsgInsert takes responsibility for destroying the message.
See Also	TipcConnMsgInsert, TipcSrvMsgNext

Examples This example constructs a message and inserts it onto the front of the message queue of the connection to RTserver:

```
T_IPC_MT mt;
T_IPC_MSG msg;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "temperature", 98.6)) {
    return; /* error */
}

if (!TipcSrvMsgInsert(msg, 0)) {
    return; /* error */
}
/* At this point, msg "belongs" to the connection to RTserver, */
/* and should not be destroyed by the caller */
```


TipcSrvMsgNext

Name	TipcSrvMsgNext — get the next message from the connection to RTserver
Synopsis	<pre>T_IPC_MSG TipcSrvMsgNext(<i>timeout</i>) T_REAL8 <i>timeout</i>;</pre>
Arguments	<i>timeout</i> — maximum number of seconds to wait for message to arrive
Return Values	A message if successful, NULL otherwise.
Diagnostics	If TipcSrvMsgNext fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgNext
Description	TipcSrvMsgNext gets the next message from the connection to RTserver by calling TipcConnMsgNext. A connection has a priority queue of incoming messages. See TipcConnMsgNext for more information on getting messages from a connection's message queue. TipcSrvMsgNext may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, and if a message is immediately available, TipcSrvMsgNext gets the next message from the warm connection. If no messages are queued in the warm connection, then TipcSrvMsgNext calls TutSleep with timeout, so that RTclient does not use an abnormal amount of CPU time. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	Call TipcMsgDestroy to destroy the message returned by TipcSrvMsgNext when it is no longer needed, unless the message is passed to another function which takes responsibility for destroying the message (such as TipcSrvMsgInsert). RTclients that receive messages with TipcSrvMsgNext or TipcSrvMsgSearch but do not call TipcSrvMsgProcess on those message are not monitorable with the TipcMon* API or RTmon. RTclient monitoring is implemented using connection process callbacks, which are all called by TipcSrvMsgProcess.
See Also	TipcConnMsgNext, TipcSrvMsgProcess, TipcSrvMainLoop, TipcSrvRead

Examples This example waits for up to 10 seconds to get the next message from the connection to RTserver, processes the message, and finally destroys the message:

```
T_IPC_MSG msg;

msg = TipcSrvMsgNext(10.0);
if (msg == NULL) {
    return; /* error */
}

if (!TipcSrvMsgProcess(msg)) {
    return; /* error */
}

if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcSrvMsgProcess

Name	TipcSrvMsgProcess — process a message in the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvMsgProcess(msg) T_IPC_MSG msg;</pre>
Arguments	<i>msg</i> — message to process
Return Values	TRUE if message was successfully processed, FALSE otherwise.
Diagnostics	If TipcSrvMsgProcess fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgProcess
Description	TipcSrvMsgProcess processes a message in the connection to RTserver by calling TipcConnMsgProcess. The process callbacks or default callbacks are executed. See TipcConnMsgProcess for more information on processing messages. TipcSrvMsgProcess may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvMsgProcess processes the message using the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	RTclients that receive messages with TipcSrvMsgNext or TipcSrvMsgSearch but do not call TipcSrvMsgProcess on those message are not monitorable with the TipcMon* API or RTmon. RTclient monitoring is implemented using connection process callbacks, which are all called by TipcSrvMsgProcess.
See Also	TipcConnMsgProcess, TipcSrvDefaultCbCreate, TipcSrvMsgNext, TipcSrvMainLoop, TipcSrvProcessCbCreate

Examples This example gets the next message from the connection to RTserver and processes it with TipcSrvMsgProcess:

```
msg = TipcSrvMsgNext(T_TIMEOUT_FOREVER);
if (msg == NULL) {
    return; /* error */
}

if (!TipcSrvMsgProcess(msg)) {
    return; /* error */
}

if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcSrvMsgSearch

Name	TipcSrvMsgSearch — search the message queue of the connection to RTserver for a specific message
Synopsis	<pre>T_IPC_MSG TipcSrvMsgSearch(<i>timeout</i>, <i>func</i>, <i>arg</i>) T_REAL8 <i>timeout</i>; T_IPC_CONN_MSG_SEARCH_FUNC <i>func</i>; T_PTR <i>arg</i>;</pre>
Arguments	<i>timeout</i> — maximum number of seconds to wait for desired message to arrive <i>func</i> — function to call once for each message <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	A message if successful, NULL otherwise.
Diagnostics	If TipcSrvMsgSearch fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgSearch
Description	TipcSrvMsgSearch searches the message queue of the connection to RTserver for a specific message by calling TipcConnMsgSearch. A connection has a priority queue of incoming messages. See TipcConnMsgSearch for more information on searching a connection's message queue. TipcSrvMsgSearch may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, and if any messages are in the warm connection message queue, TipcSrvMsgSearch searches the messages in the warm connection. If no messages are queued in the warm connection, then TipcSrvMsgSearch calls TutSleep with timeout, so that RTclient does not use an abnormal amount of CPU time. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.

Caution The message returned by `TipcSrvMsgSearch` should be destroyed when it is no longer needed by calling `TipcMsgDestroy`, unless the message is passed to another function which takes responsibility for destroying the message, such as `TipcSrvMsgInsert`.

RTclients that receive messages with `TipcSrvMsgNext` or `TipcSrvMsgSearch` but do not call `TipcSrvMsgProcess` on those message are not monitorable with the `TipcMon*` API or `RTmon`. RTclient monitoring is implemented using connection process callbacks, which are all called by `TipcSrvMsgProcess`.

See Also `TipcConnMsgSearch`, `TipcSrvMsgSearchType`

Examples This example waits for up to 10 seconds for a message to arrive that was sent to the `/system/_all` subject:

```
/* ===== */
/*..search_for_dest -- search one message */
static T_ENTRY T_BOOL search_for_dest(conn, msg, arg)
T_IPC_CONN conn;
T_IPC_MSG msg;
T_PTR arg; /* really (T_STR) */
{
    T_STR desired_dest = arg;
    T_STR dest;

    if (!TipcMsgGetDest(msg, &dest)) {
        return; /* error */
    }
    return strcmp(dest, desired_dest) == 0;
} /* search_for_dest */

/*...fragment that uses above search function */

msg = TipcSrvMsgSearch(10.0, search_for_dest, "/system/_all");
if (msg == NULL) {
    return; /* error */
}
```

TipcSrvMsgSearchType

Name	TipcSrvMsgSearchType — search the message queue of a connection to RTserver for a message with a specific type
Synopsis	<pre>T_IPC_MSG TipcSrvMsgSearchType(<i>timeout</i>, <i>mt</i>) T_REAL8 <i>timeout</i>; T_IPC_MT <i>mt</i>;</pre>
Arguments	<i>timeout</i> — maximum number of seconds to wait for desired message to arrive <i>mt</i> — desired message type
Return Values	A message if successful, NULL otherwise.
Diagnostics	If TipcSrvMsgSearchType fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgSearchType
Description	TipcSrvMsgSearchType is a convenience function that searches for a message with a specific type by calling TipcConnMsgSearchType. A connection has a priority queue of incoming messages. See TipcConnMsgSearchType for more information on searching a connection's message queue. TipcSrvMsgSearchType may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, and if any messages are in the warm connection message queue, TipcSrvMsgSearchType searches the messages in the warm connection. If no messages are queued in the warm connection, then TipcSrvMsgSearchType calls TutSleep with <i>timeout</i>, so that RTclient does not use an abnormal amount of CPU time. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	The message returned by TipcSrvMsgSearchType should be destroyed when it is no longer needed by calling TipcMsgDestroy, unless the message is passed to another function which takes responsibility for destroying the message (such as TipcSrvMsgInsert).
See Also	TipcConnMsgSearchType, TipcSrvMsgSearch

Examples This example waits indefinitely for the next CONTROL message to arrive from the connection to RTserver:

```
mt = TipcMtLookupByNum(T_MT_CONTROL);
if (mt == NULL) {
    return; /* error */
}

msg = TipcSrvMsgSearchType(T_TIMEOUT_FOREVER, mt);
if (msg == NULL) {
    return; /* error */
}
```


TipcSrvMsgSend

Name	TipcSrvMsgSend — publish (send) a message through the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvMsgSend(msg, check_server_msg_send) T_IPC_MSG msg; T_BOOL check_server_msg_send;</pre>
Arguments	<i>msg</i> — message to send <i>check_server_msg_send</i> — whether or not to check the option <code>Server_Msg_Send</code> first
Return Values	TRUE if <i>msg</i> was successfully sent, FALSE otherwise.
Diagnostics	If TipcSrvMsgSend fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> was null or the destination of <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — <i>check_server_msg_send</i> was not TRUE or FALSE, or destination of <i>msg</i> is not a valid subject • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgSend
Description	TipcSrvMsgSend sends a message for publish-subscribe through the connection to RTserver by calling TipcConnMsgSend. If <i>check_server_msg_send</i> is TRUE and the option <code>Server_Msg_Send</code> is FALSE, then TipcSrvMsgSend does not send the message, but returns TRUE immediately. Otherwise TipcSrvMsgSend first temporarily sets the sender of the message (using the value of the option <code>Unique_Subject</code>), then temporarily prepends the value of the option <code>Default_Subject_Prefix</code> to the message sender and destination if they do not start with slash (/), and then calls TipcConnMsgSend. See TipcConnMsgSend for more information on sending messages through connections. For normal usage, <i>check_server_msg_send</i> should always be TRUE. This allows the user to globally turn off all normal outgoing messages by setting the option <code>Server_Msg_Send</code> to FALSE. This is useful for running two RTclients (such as two RTie processes), the primary one with <code>Server_Msg_Send</code> set to TRUE and the backup one with <code>Server_Msg_Send</code> set to FALSE. If RTclient has a warm connection to RTserver, TipcSrvMsgSend buffers the message as if the RTclient were still connected to RTserver. See TipcSrvCreate on warm connections to RTserver.

Caution None

See Also TipcConnMsgSend, TipcSrvMsgWrite

Examples This example creates a NUMERIC_DATA message, appends fields, sets the destination subject, sends the message with TipcSrvMsgSend, sets the destination to a second subject, and sends the message again:

```
T_IPC_MT mt;
T_IPC_MSG msg;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "speed_limit", 65.0)) {
    return; /* error */
}

if (!TipcMsgSetDest(msg, "/system/admin")) {
    return; /* error */
}

/* send the message */
if (!TipcSrvMsgSend(msg, TRUE)) {
    return; /* error */
}

if (!TipcMsgSetDest(msg, "/system/test")) {
    return; /* error */
}

/* send the message again, but to a different subject */
if (!TipcSrvMsgSend(msg, TRUE)) {
    return; /* error */
}
```

TipcSrvMsgSendRpc

Name	TipcSrvMsgSendRpc — make a remote procedure call (RPC) with messages on the connection to RTserver
Synopsis	<pre>T_IPC_MSG TipcSrvMsgSendRpc(<i>call_msg</i>, <i>timeout</i>) T_IPC_MSG <i>call_msg</i>; T_REAL8 <i>timeout</i>;</pre>
Arguments	<i>call_msg</i> — call message for RPC <i>timeout</i> — maximum number of seconds to wait for result message to arrive
Return Values	A result message if successful, NULL otherwise.
Diagnostics	If TipcSrvMsgSendRpc fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>call_msg</i> was null or the destination of <i>call_msg</i> was null • T_ERR_MSG_INVALID — <i>call_msg</i> was not a valid message • T_ERR_VAL_INVALID — destination of <i>call_msg</i> is not a valid subject • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgSendRpc
Description	TipcSrvMsgSendRpc makes a remote procedure call (RPC) with messages on the connection to RTserver by calling TipcConnMsgSendRpc. One message is sent as the RPC call from the caller, and another message is sent back as the RPC result to the caller. See TipcConnMsgSendRpc for more information on RPCs that use message passing. TipcSrvMsgSendRpc may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvMsgSendRpc returns null immediately. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.

Caution When the message returned by `TipcSrvMsgSendRpc` is no longer needed, destroy it by calling `TipcMsgDestroy`, unless the message is passed to another function which takes responsibility for destroying the message, such as `TipcSrvMsgInsert`.

The message type number of the result message must be one greater than the message type number of the call message.

`TipcSrvMsgSendRpc` does not work well if multiple RTclients receive the call message, and thus all send a result message back to the caller.

`TipcSrvMsgSendRpc` returns the first result message in this case.

See Also `TipcConnMsgSendRpc`

Examples This example queries the current time of another RTclient with a message-based connection RPC. It sends a call message and waits for up to 10 seconds for a result message to be sent back from the other end of the connection. Shown first is a utility fragment that must be used by both the caller and the callee:

```
#define USER_MT_TIME_QUERY_CALL 100
#define USER_MT_TIME_QUERY_RESULT 101 /* call num plus one */

/* utility function to create message types */
static void T_ENTRY create_time_query_mt()
{
    T_IPC_MT mt;

    mt = TipcMtCreate("time_query_call",
                     USER_MT_TIME_QUERY_CALL,
                     "");

    if (mt == NULL) {
        return; /* error */
    }

    mt = TipcMtCreate("time_query_result",
                     USER_MT_TIME_QUERY_RESULT,
                     "real8");

    if (mt == NULL) {
        return; /* error */
    }
} /* create_time_query_mt */
```

Shown next is the caller fragment.

```

/* query current time via RPC */
T_REAL8 T_ENTRY time_query_rpc(unique_subject)
T_STR unique_subject; /* unique subject of remote process */
{
    T_IPC_MT mt;
    T_IPC_MSG call_msg;
    T_IPC_MSG result_msg;
    T_REAL8 result;

    mt = TipcMtLookupByNum(USER_MT_TIME_QUERY_CALL);
    if (mt == NULL) {
        return; /* error */
    }

    call_msg = TipcMsgCreate(mt);
    if (call_msg == NULL) {
        return; /* error */
    }
    if (!TipcMsgSetDest(call_msg, unique_subject)) {
        return; /* error */
    }

    /* send/recv RPC messages */
    result_msg = TipcSrvMsgSendRpc(call_msg, 10.0);
    if (result_msg == NULL) {
        return; /* error */
    }

    /* process result without using TipcSrvMsgProcess */

    if (!TipcMsgNextReal8(result_msg, &result)) {
        return; /* error */
    }
    if (!TipcMsgDestroy(result_msg)) {
        return; /* error */
    }
    if (!TipcMsgDestroy(call_msg)) {
        return; /* error */
    }
    return result;
} /* time_query_rpc */

```

This is the callee (the end opposite of the caller) fragment:

```

/* callback to process RPC "call" message */
static void T_ENTRY process_call(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR caller; /* unique subject of caller process */

    mt = TipcMtLookupByNum(USER_MT_TIME_QUERY_RESULT);
    if (mt == NULL) {
        return; /* error */
    }
}

```

```

    /* get the sender property of the message, which is the */
    /* unique subject of the caller process */
    if (!TipcMsgGetSender(data->msg, &caller)) {
        return; /* error */
    }
    /* send a result message back to the caller */
    if (!TipcSrvMsgWrite(caller, mt, FALSE,
                        T_IPC_FT_REAL8, TutGetCurrentTime(),
                        NULL)) {
        return; /* error */
    }
    if (!TipcSrvFlush()) {
        return; /* error */
    }
} /* process_call */

/*...fragment that uses process_call */
T_IPC_MT mt;

mt = TipcMtLookupByNum(USER_MT_TIME_QUERY_CALL);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, process_call, NULL) == NULL) {
    return; /* error */
}
TipcSrvMainLoop(T_TIMEOUT_FOREVER); /* process call messages */

```

TipcSrvMsgWrite

Name	TipcSrvMsgWrite — construct a message and send it through the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvMsgWrite(<i>dest</i>, <i>mt</i>, <i>check_server_msg_send</i>, ...) T_STR <i>dest</i>; T_IPC_MT <i>mt</i>; T_BOOL <i>check_server_msg_send</i>;</pre>
Arguments	<i>dest</i> — destination subject for message <i>mt</i> — message type to use for message <i>check_server_msg_send</i> — whether or not to check the option <code>Server_Msg_Send</code> first additional arguments — null-terminated field type-value pairs
Return Values	TRUE if the message was successfully constructed and sent, FALSE otherwise.
Diagnostics	If TipcSrvMsgWrite fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_NULL_PTR</code> — <i>dest</i> was null or <i>mt</i> was null • <code>T_ERR_VAL_INVALID</code> — <i>check_server_msg_send</i> was not TRUE or FALSE, or <i>dest</i> is not a valid subject • any error number from TipcSrvMsgWriteVa
Description	TipcSrvMsgWrite is a convenience function that takes a null-terminated variable number of field type-value pairs. The field type part of the pair is <code>T_IPC_FT</code>, and the value part of the pair corresponds to the arguments for the related TipcMsgAppend* function (if the type part is <code>T_IPC_FT_REAL8</code>, then the value part is a REAL8 number) or TipcMsgSet* function (if the type part is <code>T_IPC_PROP_DELIVERY_MODE</code>, then the value part is a <code>T_IPC_DELIVERY_MODE</code> setting). Some field types, such as arrays, use two arguments for the value part. TipcSrvMsgWrite performs these steps: 1. Create a message of type <i>mt</i> 2. Append the fields to the message 3. Set the message sender to the value of the option <code>Unique_Subject</code> 4. Set the message destination to <i>dest</i> 5. Send the message with TipcSrvMsgSend 6. Destroy the message with TipcMsgDestroy

TpcSrvMsgWrite is a short wrapper function that calls TpcSrvMsgWriteVa.

If *check_server_msg_send* is TRUE and the option *Server_Msg_Send* is FALSE, then TpcSrvMsgWrite does not send the message, but returns TRUE immediately. For normal usage, *check_server_msg_send* should always be TRUE. This allows the user to globally turn off all normal outgoing messages by setting the option *Server_Msg_Send* to FALSE. This is useful for running two RTclients, the primary one with *Server_Msg_Send* set to TRUE and the backup one with *Server_Msg_Send* set to FALSE.

TpcSrvMsgWrite may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TpcSrvMsgWrite buffers the message as if the RTclient were still connected to RTserver. See TpcSrvCreate on automatic creation of connections and warm connections to RTserver.

Caution None

See Also TpcConnMsgWrite, TpcSrvMsgSend, TpcSrvMsgWriteVa

Examples This example constructs and sends a NUMERIC_DATA message with TpcSrvMsgWrite:

```
T_IPC_MT mt;

mt = TpcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (!TpcSrvMsgWrite("/primary/control", mt, TRUE,
                    T_IPC_FT_STR, "voltage1",
                    T_IPC_FT_REAL8, 65.0,
                    T_IPC_FT_STR, "current1",
                    T_IPC_FT_REAL8, 10.335,
                    NULL)) {
    return; /* error */
}
```


TipcSrvMsgWriteVa

Name	TipcSrvMsgWriteVa — construct a message and send it through the connection to RTserver (<i>va_list</i> version)
Synopsis	<pre>T_BOOL TipcSrvMsgWriteVa(<i>dest</i>, <i>mt</i>, <i>check_server_msg_send</i>, <i>var_arg_list</i>) T_STR <i>dest</i>; T_IPC_MT <i>mt</i>; T_BOOL <i>check_server_msg_send</i>; va_list <i>var_arg_list</i>;</pre>
Arguments	<i>dest</i> — message destination <i>mt</i> — message type <i>check_server_msg_send</i> — whether or not to check the option <code>Server_Msg_Send</code> first <i>var_arg_list</i> — argument list
Return Values	TRUE if the message was successfully constructed and sent, FALSE otherwise.
Diagnostics	If TipcSrvMsgWriteVa fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>dest</i> was null or <i>mt</i> was null • T_ERR_VAL_INVALID — <i>check_server_msg_send</i> was not TRUE or FALSE, or <i>dest</i> is not a valid subject • any error number from TipcMsgWriteVa • any error number from TipcSrvMsgSend
Description	TipcSrvMsgWriteVa is the non-varargs version of TipcSrvMsgWrite (TipcSrvMsgWrite actually calls TipcSrvMsgWriteVa). TipcSrvMsgWriteVa can be used to implement a varargs function. In the C language, a varargs function (one that takes a variable number of arguments) cannot call another varargs function with the original arguments. A varargs function usually calls a helper function that takes a fixed number of arguments, with the last argument usually of type <i>va_list</i>. TipcSrvMsgWriteVa is TipcSrvMsgWrite's helper function.
Caution	None
See Also	TipcSrvMsgWrite, TipcMsgWriteVa, TipcSrvMsgSend

Examples This example shows how to write a varargs function in ANSI C that is similar to TipcSrvMsgWrite, but always checks the option Server_Msg_Send:

```
T_BOOL T_ENTRY my_srv_msg_write(T_STR dest,
                                T_IPC_MT mt,
                                ...)
{
    va_list var_arg_list;
    T_BOOL status;

    va_start(var_arg_list, mt);
    status = TipcSrvMsgWriteVa(dest, mt, TRUE, var_arg_list);
    va_end(var_arg_list);
    return status;
} /* my_srv_msg_write */
```

TipcSrvPrint

Name	TipcSrvPrint — print all information about the connection to RTserver in portable, reproducible format
Synopsis	<pre>T_BOOL TipcSrvPrint(<i>func</i>) T_OUT_FUNC <i>func</i>;</pre>
Arguments	<i>func</i> — TutOut-style output function
Return Values	TRUE if the connection to RTserver was successfully printed, FALSE otherwise.
Diagnostics	If TipcSrvPrint fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvPrint prints all the information about the connection to RTserver. TipcSrvPrint may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvPrint prints the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	See the <i>TIBCO SmartSockets Utilities</i> for information on TutOut
Examples	This example creates a connection to RTserver and then prints information about the connection: <pre>if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) { return; /* error */ } if (!TipcSrvPrint(TutOut)) { return; /* error */ }</pre>

The output from the above fragment would be similar to:

```

Connecting to project <rtworks> on <_node> RTserver.
Using local protocol.
Message from RTserver: Connection established.
Start subscribing to subject </_workstation1.tibco.com_8493>.
Printing connection named <client:local:workstation1.tibco.com:RTSERVER>.
logical_name = local:workstation1.tibco.com:RTSERVER.
socket_fd = 4.
read_buf: current = 0, size = 0, max = 65536
write_buf: current = 0, size = 0, max = 65536
num_queued = 0
int format = BIG_ENDIAN
real format = IEEE
char format = 437
unique subject = /_workstation1.tibco.com_8491
node:user:pid = workstation1.tibco.com:ssuser:8491
arch = sun4_solaris
block mode = false
auto flush size = 8192
read timeout = 30
write timeout = 30
keep alive timeout = 5
delivery timeout = 30
gmd file name = rtserver
auto gmd mode = true
num error callbacks = 1
no accept callbacks
num encode callbacks = 1
no decode callbacks
num global read callbacks = 1
no read callbacks
no global write callbacks
no write callbacks
no global process callbacks
num msg types with process callbacks = 16
  msg type <mon_client_msg_rcv_set_watch> has 1 callbacks.
  msg type <mon_client_buffer_poll_call> has 1 callbacks.
  msg type <control> has 1 callbacks.
  msg type <mon_client_buffer_set_watch> has 1 callbacks.
  msg type <mon_client_time_poll_call> has 1 callbacks.
  msg type <mon_client_time_set_watch> has 1 callbacks.
  msg type <mon_client_general_poll_call> has 1 callbacks.
  msg type <keep_alive_call> has 1 callbacks.
  msg type <gmd_failure> has 2 callbacks.
  msg type <gmd_nack> has 1 callbacks.
  msg type <mon_client_subject_poll_call> has 1 callbacks.
  msg type <mon_client_msg_traffic_poll_call> has 1 callbacks.
  msg type <mon_client_cb_poll_call> has 1 callbacks.
  msg type <mon_client_msg_type_poll_call> has 1 callbacks.
  msg type <mon_client_msg_send_set_watch> has 1 callbacks.
  msg type <mon_client_option_poll_call> has 1 callbacks.
no default process callbacks
no global queue callbacks
no queue callbacks

```

TipcSrvProcessCbCreate

Name	TipcSrvProcessCbCreate — create a process callback in the connection to RTserver
Synopsis	<pre>T_CB TipcSrvProcessCbCreate(<i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_MT <i>mt</i>; T_IPC_CONN_PROCESS_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvProcessCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnProcessCbCreate
Description	TipcSrvProcessCbCreate creates a process callback in the connection to RTserver by calling TipcConnProcessCbCreate. These callbacks are called by TipcSrvMsgProcess to perform the normal processing of a message. See TipcConnProcessCbCreate for more information on process callbacks. TipcSrvProcessCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvProcessCbCreate creates a process callback in the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnProcessCbCreate, TipcSrvProcessCbLookup

Examples This example creates a process callback for TIME messages in the connection to RTserver which prints the new time:

```
/*===== */
/*..my_time_process_cb -- process TIME messages */
void T_ENTRY my_time_process_cb(conn, data, arg)
T_IPC_CONN conn; /* connection to RTserver */
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_REAL8 new_time;

    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    if (!TipcMsgNextReal8(data->msg, &new_time)) {
        return; /* error */
    }

    TutOut("New time is %s\n", TutRealToStr(new_time));
} /* my_time_process_cb */

/*===== */
/*...code from calling function is below */

mt = TipcMtLookupByNum(T_MT_TIME);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvProcessCbCreate(mt, my_time_process_cb, NULL)
    == NULL) {
    return; /* error */
}
```

TipcSrvProcessCbLookup

Name	TipcSrvProcessCbLookup — look up a process callback in the connection to RTserver
Synopsis	<pre>T_CB TipcSrvProcessCbLookup(<i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_MT <i>mt</i>; T_IPC_CONN_PROCESS_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcSrvProcessCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnProcessCbLookup
Description	TipcSrvProcessCbLookup looks up the process callback with the desired message type, function, and argument in the connection to RTserver by calling TipcConnProcessCbLookup. These callbacks are called by TipcSrvMsgProcess to perform the normal processing of a message. See TipcConnProcessCbCreate for more information on connection process callbacks. TipcSrvProcessCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvProcessCbLookup looks up a process callback in the warm connection. See TipcSrvCreate for more information on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnProcessCbLookup, TipcSrvProcessCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This looks up a global process callback in the connection to RTserver:

```
T_CB cb;

cb = TipcSrvProcessCbLookup(NULL, my_global_srv_process_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
```


TipcSrvQueueCbCreate

Name	TipcSrvQueueCbCreate — create a queue callback in the connection to RTserver
Synopsis	<pre>T_CB TipcSrvQueueCbCreate(<i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_MT <i>mt</i>; T_IPC_CONN_QUEUE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvQueueCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnQueueCbCreate
Description	TipcSrvQueueCbCreate creates a queue callback in the connection to RTserver by calling TipcConnQueueCbCreate. These callbacks are called when a message is inserted into or deleted from the message queue of the connection to RTserver. See TipcConnQueueCbCreate for more information on queue callbacks. TipcSrvQueueCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvQueueCbCreate creates a queue callback in the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnQueueCbCreate, TipcSrvQueueCbLookup

Examples This example creates a global queue callback in the connection to RTserver which prints the number of queued messages:

```
void T_ENTRY my_srv_queue_cb(conn, data, arg)
T_IPC_CONN conn; /* connection to RTserver */
T_IPC_CONN_QUEUE_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_INT4 num_queued;

    if (!TipcSrvGetNumQueued(&num_queued)) {
        return; /* error */
    }

    /* print the queue size */
    TutOut("Number of queued messages from RTserver is now %d.\n",
           num_queued);
} /* my_srv_queue_cb */

/* ===== */
/*...code from main program is below */

if (TipcSrvQueueCbCreate(NULL, /* NULL signifies global */
                        my_srv_queue_cb,
                        NULL) == NULL) {
    return; /* error */
}
```

TipcSrvQueueCbLookup

Name	TipcSrvQueueCbLookup — look up a queue callback in the connection to RTserver
Synopsis	<pre>T_CB TipcSrvQueueCbLookup(<i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_MT <i>mt</i>; T_IPC_CONN_QUEUE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcSrvQueueCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnQueueCbLookup
Description	TipcSrvQueueCbLookup looks up a queue callback with the desired message type, function, and argument in the connection to RTserver by calling TipcConnQueueCbLookup. These callbacks are called when a message is inserted into or deleted from the message queue of the connection to RTserver. See TipcConnQueueCbCreate for more information on connection queue callbacks. TipcSrvQueueCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvQueueCbLookup looks up a queue callback in the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnQueueCbCreate, TipcSrvQueueCbCreate, TipcConnQueueCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example looks up and destroys a NUMERIC_DATA queue callback in the connection to RTserver:

```
T_CB cb;
T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

cb = TipcSrvQueueCbLookup(mt, my_numeric_queue_cb, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}
```

TipcSrvRead

Name	TipcSrvRead — read all available data from the connection to RTserver and queue messages in priority order
Synopsis	<pre>T_BOOL TipcSrvRead(<i>timeout</i>) T_REAL8 <i>timeout</i>;</pre>
Arguments	<i>timeout</i> — maximum number of seconds to wait for data to arrive
Return Values	TRUE if data was successfully read from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvRead fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnRead
Description	TipcSrvRead reads and queues all available messages from the connection to RTserver by calling TipcConnRead. See TipcConnRead for more information on reading data from a connection. TipcSrvRead may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvRead sleeps for <i>timeout</i> seconds and returns FALSE. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	TipcSrvRead is a low-level function that is normally not used directly by developers; TipcSrvRead is normally called only by TipcSrvMsgNext and TipcSrvMsgSearch.
See Also	TipcConnRead, TipcSrvCheck, TipcSrvReadCbCreate
Examples	This example reads all data that is immediately available from the connection to RTserver with TipcSrvRead: <pre>if (!TipcSrvRead(0.0)) { return; /* error */ }</pre>

TipcSrvReadCbCreate

Name	TipcSrvReadCbCreate — create a read callback in the connection to RTserver
Synopsis	<pre>T_CB TipcSrvReadCbCreate(mt, func, arg) T_IPC_MT mt; T_IPC_CONN_READ_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvReadCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnReadCbCreate
Description	TipcSrvReadCbCreate creates a read callback in the connection to RTserver by calling TipcConnReadCbCreate. These callbacks are called when a message is read from the connection to RTserver. See TipcConnReadCbCreate for more information on read callbacks. TipcSrvReadCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvReadCbCreate creates a read callback in the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnReadCbCreate, TipcSrvReadCbLookup

Examples

This example creates a read callback in the connection to RTserver for TIME messages which compares the current data time to the data time in the message. If the two times differ by more than 100, a warning is printed, which provides a simple way for an RTclient to detect that it is falling behind:

```
/* ===== */
/*..my_srv_read_cb -- compare TIME message against current time */
void T_ENTRY my_srv_read_cb(conn, data, arg)
T_IPC_CONN conn; /* connection to RTserver */
T_IPC_CONN_READ_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_IPC_MSG msg = data->msg;
    T_REAL8 new_time;

    if (!TipcMsgSetCurrent(msg, 0)) {
        return; /* error */
    }

    if (!TipcMsgNextReal8(msg, &new_time)) {
        return; /* error */
    }

    /* compare the new time to the current time */
    if (new_time - TutGetCurrentTime() > 100.0) {
        TutWarning("We appear to be %s seconds behind!\n",
            TutRealToStr(new_time - TutGetCurrentTime()));
    }
} /* my_srv_read_cb */

/* ===== */
/*...code from main program is below */

mt = TipcMtLookupByNum(T_MT_TIME);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvReadCbCreate(mt, my_srv_read_cb, NULL) == NULL) {
    return; /* error */
}
```

TipcSrvReadCbLookup

Name	TipcSrvReadCbLookup — look up a read callback in the connection to RTserver
Synopsis	<pre>T_CB TipcSrvReadCbLookup(<i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_MT <i>mt</i>; T_IPC_CONN_READ_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcSrvReadCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnReadCbLookup
Description	TipcSrvReadCbLookup looks up a read callback with the desired message type, function, and argument in the connection to RTserver by calling TipcConnReadCbLookup. These callbacks are called when a message is read from the connection to RTserver. See TipcConnReadCbCreate on connection read callbacks. TipcSrvReadCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvReadCbLookup looks up a read callback in the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnReadCbCreate, TipcSrvReadCbCreate, TipcSrvReadCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This code looks up and destroys a global read callback in the connection to RTserver:

```
T_CB cb;

cb = TipcSrvReadCbLookup(NULL, my_global_read_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
if (!TutCbDestroy(cb)) {
    return; /* error */
}
```

TipcSrvSetAutoFlushSize

Name	TipcSrvSetAutoFlushSize — set the auto flush size of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvSetAutoFlushSize(<i>auto_flush_size</i>) T_INT4 <i>auto_flush_size</i>;</pre>
Arguments	<i>auto_flush_size</i> — auto flush size (in bytes)
Return Values	TRUE if the auto flush size was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvSetAutoFlushSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_TOO_SMALL — <i>auto_flush_size</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnSetAutoFlushSize
Description	TipcSrvSetAutoFlushSize sets the auto flush size of the connection to RTserver by calling TipcConnSetAutoFlushSize. The auto flush size is used to control how much outgoing data (in bytes) is buffered before being automatically flushed to RTserver. The default for the auto flush size is 8192 bytes. See TipcConnGetAutoFlushSize for more information on auto flush sizes. TipcSrvSetAutoFlushSize may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvSetAutoFlushSize sets the auto flush size of the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	Unlike with other connections, the auto flush size T_IPC_NO_AUTO_FLUSH_SIZE cannot be used with the connection to RTserver to disable all automatic flushing of outgoing messages.
See Also	TipcConnGetAutoFlushSize, TipcConnSetAutoFlushSize, TipcSrvGetAutoFlushSize

Examples This example creates a connection to RTserver, then sets the auto flush size to 4096 bytes:

```
if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {  
    return; /* error */  
}  
  
if (!TipcSrvSetAutoFlushSize(4096)) {  
    return; /* error */  
}
```

TipcSrvSetBlockMode

Name	TipcSrvSetBlockMode — set the block mode of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvSetBlockMode(<i>block_mode</i>) T_BOOL <i>block_mode</i>;</pre>
Arguments	<i>block_mode</i> — block mode
Return Values	TRUE if block mode was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvSetBlockMode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>block_mode</i> was not TRUE or FALSE • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnSetBlockMode
Description	TipcSrvSetBlockMode sets the block mode of the connection to RTserver by calling TipcConnSetBlockMode. The block mode controls whether or not the RTclient blocks waiting for read and write operations to complete. See TipcConnGetBlockMode for more information on connection block modes. TipcSrvSetBlockMode may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvSetBlockMode returns FALSE immediately. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	Use TipcSrvSetBlockMode with caution. The block mode of the connection to RTserver is automatically set to FALSE if both of the options Server_Read_Timeout and Server_Write_Timeout are non-zero. This is needed because the timeouts cannot take effect if the block mode is TRUE. In a similar fashion, the block mode of the connection to RTserver is automatically set to TRUE if either of the options Server_Read_Timeout or Server_Write_Timeout are zero.
See Also	TipcConnSetBlockMode, TipcSrvGetBlockMode
Examples	This example sets the block mode of the connection to RTserver to FALSE so that the RTclient does not block waiting for read and write operations to complete: <pre>if (!TipcSrvSetBlockMode(FALSE)) { return; /* error */ }</pre>

TipcSrvSetCredentials

Name	TipcSrvSetCredentials — set the credentials of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvSetCredentials(auth_policy_id, auth_data, auth_data_len) T_INT4 auth_policy_id; T_PTR auth_data; T_INT4 auth_data_len;</pre>
Arguments	<i>auth_policy_id</i> — authentication policy identifier <i>auth_data</i> — authentication data specified by the authentication policy identifier <i>auth_data_len</i> — size of the authentication data (in bytes)
Return Values	TRUE if the credentials were successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvSetCredentials fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>auth_data</i> was null • T_ERR_VAL_TOO_SMALL — <i>auth_data_len</i> was negative • T_ERR_SRV_ALREADY_CONNECTED — a full connection to RTserver was already established
Description	TipcSrvSetCredentials sets the credentials of the connection to RTserver. The credentials are used to authenticate the connection. If authentication fails an appropriate error code is set when TipcSrvCreate is called. Calling TutErrNumGet returns the error code.
Caution	None
See Also	TipcSrvCreate

Examples This example sets the RTclient's credentials and attempts to make a connection to the RTserver:

```
#define AUTH_BASIC 1

T_STR user_pass = "jdoe:387203";

if (!TipcSrvSetCredentials(AUTH_BASIC, user_pass,
    strlen(user_pass)) {
    return; /* error */
}

if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}
```

TipcSrvSetGmdMaxSize

Name	TipcSrvSetGmdMaxSize — set the GMD area maximum size of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvSetGmdMaxSize(<i>gmd_max_size</i>) T_UINT4 <i>gmd_max_size</i>;</pre>
Arguments	<i>gmd_max_size</i> — connection to RTserver GMD area maximum size (in bytes)
Return Values	TRUE if the GMD area maximum size was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvSetGmdMaxSize fails, it returns FALSE and sets the global SmartSockets error number to one of: - T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created - any error number from TipcConnSetGmdMaxSize
Description	TipcSrvSetGmdMaxSize sets the GMD area maximum size in bytes of the connection to RTserver by calling TipcConnSetGmdMaxSize. A GMD area holds guaranteed message delivery information for incoming and outgoing messages. There are two types of guaranteed message delivery: file-based and memory-only. See TipcConnGmdFileCreate for more information on GMD areas. For file-based GMD this limit is the maximum file size, and for memory-only GMD this limit is the maximum amount of memory allowed. The default GMD area maximum size is 0, which means that no GMD area maximum size limit checking is performed. The connection GMD area maximum size and the connection delivery timeout can be used to constrain the amount of system resources used for GMD by space or by time. If the connection GMD area maximum size is exceeded, no further messages can be sent with guaranteed delivery (TipcSrvMsgSend returns FALSE and sets the global SmartSockets error number to T_ERR_VAL_TOO_LARGE) until some unacknowledged previously sent messages are acknowledged. TipcSrvSetGmdMaxSize may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvSetGmdMaxSize sets the GMD area maximum size of the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnSetGmdMaxSize, TipcSrvGmdFileCreate, TipcSrvGetGmdMaxSize

Examples This example sets the GMD area maximum size of the connection to RTserver to one million bytes:

```
if (!TipcSrvSetGmdMaxSize(1000000)) {  
    return; /* error */  
}
```


TipcSrvSetSocket

Name	TipcSrvSetSocket — set the socket of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvSetSocket(socket) T_INT4 socket;</pre>
Arguments	<i>socket</i> — connection socket
Return Values	TRUE if socket was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvSetSocket fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnSetSocket
Description	TipcSrvSetSocket sets the socket file descriptor of the connection to RTserver by calling TipcConnSetSocket. All data sent to and received from the connection is transmitted on this socket. See TipcSrvGetSocket for more information on connection sockets. TipcSrvSetSocket may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvSetSocket does not use the warm connection, because the warm connection does not have a valid socket. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnSetSocket, TipcSrvGetSocket
Examples	This example sets the socket of the connection to RTserver: <pre>my_socket = my_create_override_socket(); if (!TipcSrvSetSocket(my_socket)) { return; /* error */ }</pre>

TipcSrvSetTimeout

Name	TipcSrvSetTimeout — set a timeout property of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvSetTimeout(<i>timeout</i>, <i>value</i>) T_IPC_TIMEOUT <i>timeout</i>; T_REAL8 <i>value</i>;</pre>
Arguments	<i>timeout</i> — which timeout property to set <i>value</i> — <i>timeout</i> value (in seconds)
Return Values	TRUE if <i>timeout</i> was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvSetTimeout fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>timeout</i> was not a valid T_IPC_TIMEOUT • T_ERR_VAL_TOO_SMALL — <i>value</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvSetTimeout sets a timeout property of the connection to RTserver by setting one of the options: Server_Read_Timeout, Server_Write_Timeout, Server_Keep_Alive_Timeout, and Server_Delivery_Timeout. The timeout properties control how often the connection to RTserver is checked for possible network failures. See TipcSrvGetTimeout for more information on connection timeouts. TipcSrvSetTimeout may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvSetTimeout sets a timeout property of the warm connection. See TipcSrvCreate for information on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnSetTimeout, TipcSrvGetTimeout

Examples This sets the read timeout of the connection to RTserver with TipcSrvSetTimeout to 10 seconds:

```
if (!TipcSrvSetTimeout(T_IPC_TIMEOUT_READ, 10.0)) {  
    return; /* error */  
}
```

An alternate way to set the read timeout of the connection to RTserver is to set the option `Server_Read_Timeout`.

```
TutCommandParseStr("setopt server_read_timeout 10.0");
```

TipcSrvSetUsernamePassword

Name	TipcSrvSetUsernamePassword — set the basic credentials of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvSetUsernamePassword(<i>username</i>, <i>password</i>) T_STR <i>username</i>; T_STR <i>password</i>;</pre>
Arguments	<i>username</i> — username to use for basic credentials <i>password</i> — password to use for basic credentials
Return Values	TRUE if the basic credentials were successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvSetUsernamePassword fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>username</i> or <i>password</i> was null • T_ERR_SRV_ALREADY_CONNECTED — a connection to RTserver has already been established
Description	TipcSrvSetUsernamePassword sets the basic credentials of the connection to RTserver. When RTserver has enabled Basic Security, the basic credentials are used to authenticate the connection. If authentication fails an error code is set when TipcSrvCreate is called. For more information on Basic Security, see the <i>TIBCO SmartSockets User’s Guide</i>.
See Also	TipcSrvCreate
Examples	This example sets the RTclient’s basic credentials and attempts to make a connection to the RTserver: <pre>if (!TipcSrvSetUsernamePassword(argv[1], argv[2])) { return; /* error */ } if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) { return; /* error */ }</pre>

TipcSrvStdSubjectSetSubscribe

Name	TipcSrvStdSubjectSetSubscribe — start or stop subscribing to all standard subjects
Synopsis	<pre>T_BOOL TipcSrvStdSubjectSetSubscribe(<i>subscribe_status</i>, <i>time_status</i>) T_BOOL <i>subscribe_status</i>; T_BOOL <i>time_status</i>;</pre>
Arguments	<i>subscribe_status</i> — whether to start or stop subscribing to all standard subjects <i>time_status</i> — whether or not to include <code>_time</code> subject
Return Values	TRUE if the subscribe statuses of the standard subjects were successfully set, FALSE otherwise.
Diagnostics	If TipcSrvStdSubjectSetSubscribe fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_VAL_INVALID</code> — <i>subscribe_status</i> or <i>time_status</i> was not TRUE or FALSE • <code>T_ERR_SRV_NOT_CONNECTED</code> — a connection to RTserver could not be created
Description	TipcSrvStdSubjectSetSubscribe sets the subscribe statuses of the standard subjects by calling TipcSrvSubjectSetSubscribe for each standard subject. TipcSrvStdSubjectSetSubscribe is often used to subscribe to subjects listed in a command file. There are many standard subjects: • <code>_time</code> — only set if <i>time_status</i> is TRUE • subjects listed in the option Subjects • <code>_node</code> — the node-specific subject, such as <code>workstation1</code> • <code>_all</code> — common subject for all RTclients • standard process type subject, such as <code>_mon</code>. User-defined RTclients do not have a standard process type subject. RTmon uses <code>_mon</code>. As each standard subject subscribe status is changed, TipcSrvStdSubjectSetSubscribe prints out an informational message with TutOut. TipcSrvStdSubjectSetSubscribe also flushes the retrieval messages with TipcSrvFlush when it is done.

TpcSrvStdSubjectSetSubscribe may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TpcSrvStdSubjectSetSubscribe changes the subscribe statuses of the subjects as if the RTclient were still connected to RTserver. See TpcSrvCreate on automatic creation of connections and warm connections to RTserver.

Caution None

See Also TpcSrvSubjectSetSubscribe

Examples This example sets the option Subjects and uses TpcSrvStdSubjectSetSubscribe to start subscribing to the standard subjects:

```
TutCommandParseStr("setopt subjects subject1, subject2");
if (!TpcSrvStdSubjectSetSubscribe(TRUE, TRUE)) {
    return; /* error */
}
```

The output from the above fragment would be similar to:

```
Start subscribing to subject </_time>.
Start subscribing to subject </subject1>.
Start subscribing to subject </subject2>.
Start subscribing to subject </_workstation1>.
Start subscribing to subject </_all>.
```

TipcSrvStop

Name	TipcSrvStop — stop RTserver and possibly more processes
Synopsis	<pre>T_BOOL TipcSrvStop(stop_type) T_IPC_SRV_STOP_TYPE stop_type;</pre>
Arguments	<i>stop_type</i> — how to stop RTserver (T_IPC_SRV_STOP_NORMAL, T_IPC_SRV_STOP_ALL, T_IPC_SRV_STOP_CLIENTS, or T_IPC_SRV_STOP_SERVERS)
Return Values	TRUE if RTserver was successfully stopped, FALSE otherwise.



For stop types other than T_IPC_SRV_STOP_NORMAL, there is no way to determine if additional clients/servers to which the message was forwarded accepted or rejected the request.

Diagnostics	If TipcSrvStop fails, it returns FALSE and sets the global SmartSockets error to one of: • T_ERR_VAL_INVALID — <i>stop_type</i> was not a valid T_IPC_SRV_STOP_TYPE. • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created. • T_ERR_SRV_CTLMSG_REJECTED — the server did not stop because the <i>enable_stop_msgs</i> option was set to false. If the stop mode is not T_IPC_SRV_STOP_NORMAL, the message is forwarded to clients/servers, which may or may not reject the message. • any error number from TipcSrvMsgSendRpc.

Description	TipcSrvStop stops RTserver and possibly more processes. TipcSrvStop uses TipcSrvMsgSendRpc to send a SERVER_STOP_CALL message with one field containing <i>stop_type</i> and then waits for RTserver to send back a SERVER_STOP_RESULT message with two fields field containing a status string and a result code. TipcSrvStop prints this status string with TutOut and, if the result is not success, sets the global SmartSockets error to the result code returned by the server.

If *stop_type* is T_IPC_SRV_STOP_NORMAL, then TipcSrvStop stops only the one RTserver that the RTclient is connected to. The RTclients that are connected to that RTserver continue to run, eventually detect that RTserver has stopped, and try to find or start a new RTserver.

If *stop_type* is `T_IPC_SRV_STOP_ALL`, then `TipcSrvStop` stops the one RTserver that the RTclient is connected to, all RTservers connected to that RTserver, and all RTclients connected to those RTservers. The RTclients are stopped by sending them a `CONTROL` message (with a destination subject of `_all`) containing the command `quit force`.

If *stop_type* is `T_IPC_SRV_STOP_CLIENTS`, then `TipcSrvStop` stops the one RTserver and all RTclients connected to that RTserver. The RTclients are stopped by sending them a `CONTROL` message (with a destination subject of `_all`) containing the command `quit force`.

If *stop_type* is `T_IPC_SRV_STOP_SERVERS`, then `TipcSrvStop` stops the one RTserver and all RTservers connected to that RTserver. The RTclients that are connected to those RTservers continue to run, eventually detect that RTserver has stopped, and try to find or start a new RTserver.

`TipcSrvStop` may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, `TipcSrvStop` returns `FALSE` immediately. See `TipcSrvCreate` on automatic creation of connections and warm connections to RTserver.

For information on the `rtserver` shell script, see the *TIBCO SmartSockets Utilities*.

Caution The `SERVER_STOP_CALL` message sent from `TipcSrvStop` will not be executed by RTservers receiving the message if the setting of the RTserver option `Enable_Stop_Msgs` (default `TRUE`) is set to `FALSE`. The return code of `TipcSrvStop` indicates success or failure only for the primary receiving server.

See Also `TipcSrvCreate`, `TipcSrvIsRunning`, `TipcSrvMsgSendRpc`; see the *TIBCO SmartSockets User's Guide* for information on the `rtserver` shell script.

Examples This example creates a connection to RTserver, then stops RTserver and all RTclients connected to that RTserver:

```
if (!TipcSrvCreate(T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}

if (!TipcSrvStop(T_IPC_SRV_STOP_CLIENTS)) {
    return; /* error */
}
```

The output from the above fragment would be similar to:

```
Connecting to project <rtworks> on <_node> RTserver.
Using local protocol.
Message from RTserver: Connection established.
Start subscribing to subject </_workstation1_7050>.
Sending RTserver stop message.
Message from RTserver: RTserver and its clients stopping.
```


TipcSrvSubjectCbCreate

Name	TipcSrvSubjectCbCreate — create a subject callback
Synopsis	<pre>T_CB TipcSrvSubjectCbCreate(<i>subject</i>, <i>mt</i>, <i>func</i>, <i>arg</i>) T_STR <i>subject</i>; T_IPC_MT <i>mt</i>; T_IPC_SRV_SUBJECT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>subject</i> — subject to create callback for (null means all subjects) <i>mt</i> — message type to create callback for (null means all message types) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvSubjectCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvSubjectCbCreate creates a subject callback. These callbacks are called by TipcSrvMsgProcess to process a message when it is received by the <i>subject</i>. If <i>subject</i> is null, then <i>func</i> is called for all messages of type <i>mt</i> received by this RTclient. If <i>mt</i> is null, then <i>func</i> is called for every type of message received by <i>subject</i>. If both <i>subject</i> and <i>mt</i> are null, <i>func</i> is called for every message received by this RTclient. Subject wildcards (* or ...) are supported.
Caution	Do not use both subject callbacks and connection process callbacks at the same time. Each callback type represents a different way of partitioning data and they are not orthogonal. Connection process callbacks partition the data according to message type; subject callbacks partition according to subjects. If these types of callbacks are used together, connection process callbacks are always of higher priority. Connection process callbacks are called in priority order until the last connection process callback is called. Then the subject callbacks are called in priority order. The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.

See Also TipcSrvSubjectCbLookup, TipcSrvSubjectDefaultCbCreate, TipcSrvSubjectDefaultCbLookup; see the *TIBCO SmartSockets Utilities* for information on TutCbSetPriority.

Examples This example creates a subject callback for T_MT_INFO messages that are received by the /stocks subject. A second subject callback is created for all message types received by the /emergency subject. The emergency callback has a higher priority so that it will be called first:

```
/* ===== */
/*..info_subject_cb -- process INFO messages recvd by /stocks */
void T_ENTRY info_subject_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_SRV_SUBJECT_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_STR info_str;
    T_STR subject;

    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    if (!TipcMsgNextStr(data->msg, &info_str)) {
        return; /* error */
    }

    if (!TipcMsgGetDest(data->msg, &subject)) {
        return; /* error */
    }

    TutOut("Information message received by %s is %s\n", subject,
info_str);
} /* info_subject_cb */

/* ===== */
/*..emergency_cb -- process messages recvd by emergency subject */
void T_ENTRY emergency_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_SRV_SUBJECT_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_STR mt_name;
    T_IPC_MT mt;

    if (!TipcMsgGetType(data->msg, &mt)) {
        return; /* error */
    }
}
```

```

    if (!TipcMtGetName(mt, &mt_name) {
        return; /* error */
    }

    TutOut("WARNING: Emergency message received: %s\n", mt_name);
} /* emergency_cb */

/* ===== */
/*...code from calling function is below */

mt = TipcMtLookupByNum(T_MT_INFO);
if (mt == NULL) {
    return; /* error */
}
cb = TipcSrvSubjectCbCreate("/stocks", mt, info_subject_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
cb = TipcSrvSubjectCbCreate("/emergency", NULL, emergency_cb,
NULL);
if (cb == NULL) {
    return; /* error */
}
if (!TutCbSetPriority(cb, 100)) {
    return; /* error */
}
}

```

TipcSrvSubjectCbDestroyAll

Name	TipcSrvSubjectDestroyAll — destroy all subject callbacks
Synopsis	T_BOOL TipcSrvSubjectCbDestroyAll(void)
Arguments	None
Return Values	TRUE if successful, FALSE otherwise.
Diagnostics	If TipcSrvSubjectCbDestroyAll fails, it returns NULL and sets the global SmartSockets error number to: T_ERR_DOESNT_EXIST — no subject callbacks exist to destroy
Description	TipcSrvSubjectCbDestroy destroys all registered subject callbacks including default subject callbacks. This is a convenience function that allows for all of the subject callbacks to be destroyed easily.
Caution	None
See Also	TipcSrvSubjectCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example destroys all subject callbacks:<pre>if (!TipcSrvSubjectCbDestroyAll()) { return; /* error */ }</pre>

TipcSrvSubjectCbLookup

Name	TipcSrvSubjectCbLookup — look up a subject callback
Synopsis	<pre>T_CB TipcSrvSubjectCbLookup(<i>subject</i>, <i>mt</i>, <i>func</i>, <i>arg</i>) T_STR <i>subject</i>; T_IPC_MT <i>mt</i>; T_IPC_SRV_SUBJECT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>subject</i> — subject to look up callback for <i>mt</i> — message type to look up callback for <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcSrvSubjectCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvSubjectCbLookup looks up a subject callback with the desired subject, message type, function, and argument. When the subject contains wildcards, the wildcarded subject must exactly match the subject that was used when creating the subject callback. If successful, TipcSrvSubjectCbLookup returns one and only one subject callback.
Caution	None
See Also	TipcSrvSubjectCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up a global subject callback for the /emergency subject: <pre>cb = TipcSrvSubjectCbLookup("/emergency", NULL, emergency_cb, NULL); if (cb == NULL) { return; /* error */ }</pre>

TipcSrvSubjectDefaultCbCreate

Name	TipcSrvSubjectDefaultCbCreate — create a default subject callback
Synopsis	<pre>T_CB_TipcSrvSubjectDefaultCbCreate(func, arg) T_IPC_SRV_SUBJECT_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvSubjectDefaultCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvSubjectDefaultCbCreate creates a default subject callback. Callbacks are called by TipcSrvMsgProcess to perform the processing of a message. The callback function, <i>func</i> , is only called when no other subject callback matches the subject of the message being processed.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcSrvSubjectCbCreate, TipcSrvSubjectCbLookup, TipcSrvSubjectDefaultCbLookup

Examples This example creates a default subject callback:

```
/* ===== */
/*..default_subject_cb -- default subject callback */
void T_ENTRY default_subject_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_SRV_SUBJECT_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_STR mt_name;
    T_IPC_MT mt;
    T_STR subject;

    if (!TipcMsgGetType(data->msg, &mt)) {
        return; /* error */
    }

    if (!TipcMtGetName(mt, &mt_name)) {
        return; /* error */
    }

    if (!TipcMsgGetDest(data->msg, &subject)) {
        return; /* error */
    }

    TutOut("DEFAULT: Message type %s recvd by %s\n", mt_name,
subject);
} /* default_subject_cb */

/* ===== */
/*...code from calling function is below */

T_CB cb;

cb = TipcSrvSubjectDefaultCbCreate(default_subject_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
```

TpcSrvSubjectDefaultCbLookup

Name	TpcSrvSubjectDefaultCbLookup — look up a default subject callback
Synopsis	<pre>T_CB TpcSrvSubjectDefaultCbLookup(<i>func</i>, <i>arg</i>) T_IPC_SRV_SUBJECT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TpcSrvSubjectDefaultCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TpcSrvSubjectDefaultCbLookup looks up a default subject callback with the desired function and argument. See TpcSrvSubjectDefaultCbCreate for more information on default subject callbacks.
Caution	None
See Also	TpcSrvSubjectDefaultCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up a default subject callback: <pre>cb = TpcSrvSubjectDefaultCbLookup(default_subject_cb, NULL); if (cb == NULL) { return; /* error */ }</pre>

TipcSrvSubjectGetSubscribe

Name	TipcSrvSubjectGetSubscribe — determine if an RTclient is subscribing to a subject
Synopsis	<pre>T_BOOL TipcSrvSubjectGetSubscribe(subject, subscribe_status_return) T_STR subject; T_BOOL *subscribe_status_return;</pre>
Arguments	<i>subject</i> — name of the subject to get subscribe status for <i>subscribe_status_return</i> — storage for subscribe status
Return Values	TRUE if the subscribe status of the subject was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcSrvSubjectGetSubscribe fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>subject</i> was null or <i>subscribe_status_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvSubjectGetSubscribe gets whether an RTclient is subscribing to a subject. The subscribe status controls whether the RTclient receives messages published (sent) to that subject. If the subscribe status of <i>subject</i> is TRUE, the RTclient receives messages with <i>subject</i> as their destination that are sent to RTserver. To start or stop subscribing to a subject, use TipcSrvSubjectSetSubscribe. To find out if an RTclient is subscribing to a subject, including load balancing information, use TipcSrvSubjectGetSubscribeLb. TipcSrvSubjectGetSubscribe may attempt to automatically create a connection to RTserver. TipcSrvSubjectGetSubscribe gets the subscribe status of the subject for both full and warm RTclient connections. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvSubjectGetSubscribe returns FALSE, it does not store a value in <i>subscribe_status_return</i> .
See Also	TipcMonClientSubscribeSetWatch, TipcMonSubjectSubscribeSetWatch, TipcSrvSubjectSetSubscribe, TipcSrvSubjectGetSubscribeLb

Examples This example gets the subscribe status of the subject /system/thermal:

```
T_BOOL subscribe_status;
if (!TipcSrvSubjectGetSubscribe("/system/thermal",
&subscribe_status)) {
    return; /* error */
}
TutOut("This process is %s the /system/thermal subject.\n",
        subscribe_status ? "subscribing to" : "not subscribing to");
```

TipcSrvSubjectGetSubscribeLb

Name	TipcSrvSubjectGetSubscribeLb — determine if an RTclient is subscribing to a subject, including load balancing information
Synopsis	<pre>T_BOOL TipcSrvSubjectGetSubscribeLb(<i>subject</i>, <i>subscribe_status_return</i>, <i>lb_status_return</i>) T_STR <i>subject</i>; T_BOOL *<i>subscribe_status_return</i>; T_BOOL *<i>lb_status_return</i>;</pre>
Arguments	<i>subject</i> — name of the subject to get subscribe status for <i>subscribe_status_return</i> — storage for subscribe status <i>lb_status_return</i> — storage for load balancing status
Return Values	TRUE if the subscribe statuses of the subject were successfully retrieved, FALSE otherwise.
Diagnostics	If TipcSrvSubjectGetSubscribeLb fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>subject</i>, <i>subscribe_status_return</i>, or <i>lb_status_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvSubjectGetSubscribeLb and TipcSrvSubjectSetSubscribeLb are variants of TipcSrvSubjectGetSubscribe and TipcSrvSubjectSetSubscribe that include load balancing control. TipcSrvSubjectGetSubscribeLb gets whether or not an RTclient is subscribing to a subject, including load balancing information. The subscribe status controls whether the RTclient receives messages published (sent) to that subject. If the subscribe status of subject is TRUE, then the RTclient receives messages with subject as their destination that are sent to RTserver. To start or stop subscribing to a subject, with or without load balancing, use TipcSrvSubjectSetSubscribeLb. If the load balancing status of subject is TRUE, then the RTclient is part of the load balancing calculation, otherwise the RTclient overrides load balancing and always receives the messages regardless of whether or not the message was published with load balancing.

TpcSrvSubjectGetSubscribeLb may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TpcSrvSubjectGetSubscribeLb gets the subscribe status of the subject as if the RTclient were still connected to RTserver. See TpcSrvCreate on automatic creation of connections and warm connections to RTserver.

Caution If TpcSrvSubjectGetSubscribeLb returns FALSE, it does not store a value in *subscribe_status_return* or *lb_status_return*.

See Also TpcMonClientSubscribeSetWatch, TpcMonSubjectSubscribeSetWatch, TpcSrvSubjectGetSubscribe, TpcSrvSubjectSetSubscribeLb

Examples This example uses TpcSrvSubjectGetSubscribeLb to get the subscribe status of the subject /system/thermal:

```
T_BOOL subscribe_status;
T_BOOL lb_status;

if (!TpcSrvSubjectGetSubscribeLb("/system/thermal",
                                &subscribe_status, &lb_status)) {
    return; /* error */
}

TutOut("This process is %s the /system/thermal subject.\n",
       subscribe_status ? "subscribing to" : "not subscribing to");

if (subscribe_status) {
    TutOut("This process is %s load balancing.\n",
          lb_status ? "part of" : "not part of");
}
```

TipcSrvSubjectGmdInit

Name	TipcSrvSubjectGmdInit — initialize GMD accounting in RTserver for a subject to which messages will be published
Synopsis	<pre>T_BOOL TipcSrvSubjectGmdInit(<i>subject</i>) T_STR <i>subject</i>;</pre>
Arguments	<i>subject</i> — name of subject to which messages will be published with GMD
Return Values	TRUE if the request for GMD initialization was successfully initiated, FALSE otherwise.
Diagnostics	If TipcSrvSubjectGmdInit fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>subject</i> was null • any error number from TipcSrvMsgWrite
Description	TipcSrvSubjectGmdInit initializes GMD accounting in RTserver for a subject to which messages will be published by sending a GMD_INIT_CALL message to RTserver. RTserver responds by sending back a GMD_INIT_RESULT message when GMD initialization is complete. The response should come back very quickly. The GMD_INIT_RESULT message contains one field: a STR field containing the subject name for which initialization has completed. The polling RTclient can use functions such as TipcSrvMsgSearchType to get the response. Both guaranteed message delivery and load balancing currently use the same accounting mechanism. Use TipcSrvSubjectLbInit, however, to initialize load balancing accounting for a subject. When the RTserver connected to the publishing RTclient routes a message for GMD, it keeps track of which RTclients should receive the message and the status of GMD to each of these RTclients. If this RTserver is not already tracking GMD for subject when the first published message arrives from RTclient, the RTserver must start a monitoring-like watch mechanism between the RTservers of all subscribing RTclients. This increases the scalability of GMD because only the relevant RTservers dynamically exchange GMD information. The GMD accounting continues until no messages have been published with GMD or load balancing for an interval specified by the Gmd_Publish_Timeout option. The default for Gmd_Publish_Timeout is 300 seconds. The starting of this GMD accounting is very fast, but not instantaneous. TipcSrvSubjectGmdInit is intended to be used by advanced publish-subscribe applications that wish to ensure that RTserver is primed and immediately ready for GMD publish-subscribe routing.

TpcSrvSubjectGmdInit is usually only used at initialization, but it can be called at any time by advanced applications that wish to initialize GMD publishing accounting. The use of TpcSrvSubjectGmdInit is strictly optional for GMD publishing; if it is not called, the GMD setup occurs when the first GMD message is published to *subject*.

If RTserver is already tracking GMD for *subject*, RTserver immediately returns a GMD_INIT_RESULT message.

Caution TpcSrvSubjectGmdInit sends a GMD_INIT_CALL message to RTserver, but does not explicitly flush the message. See TpcSrvGetAutoFlushSize for more information on message buffering.

See Also TpcSrvMsgSend, TpcSrvMsgWrite, TpcSrvSubjectLbInit

Examples This example uses TpcSrvSubjectGmdInit to initialize GMD accounting in RTserver for the /system/thermal subject and then waits for the result message:

```
T_IPC_MT mt;
T_IPC_MSG result_msg;

/* initialize GMD accounting in RTserver */
if (!TpcSrvSubjectGmdInit("/system/thermal")) {
    return; /* error */
}

mt = TpcMtLookupByNum(T_MT_GMD_INIT_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait a short time for the GMD init result */
result_msg = TpcSrvMsgSearchType(10.0, mt);
if (result_msg == NULL) {
    return; /* error */
}

/* must set current field first */
if (!TpcMsgSetCurrent(result_msg, 0)) {
    return; /* error */
}

/* at this point, we could read the first field of the result message to verify the subject name */
```

TipcSrvSubjectLbInit

Name	TipcSrvSubjectLbInit — initialize load balancing accounting in RTserver for a subject to which messages will be published
Synopsis	<pre>T_BOOL TipcSrvSubjectLbInit(<i>subject</i>) T_STR <i>subject</i>;</pre>
Arguments	<i>subject</i> — name of subject to which messages will be published with load balancing
Return Values	TRUE if the request for load balancing initialization was successfully initiated, FALSE otherwise.
Diagnostics	If TipcSrvSubjectLbInit fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>subject</i> was null • any error number from TipcSrvMsgWrite
Description	TipcSrvSubjectLbInit initializes load balancing accounting in RTserver for a subject to which messages will be published by sending a GMD_INIT_CALL message to RTserver. RTserver responds by sending back a GMD_INIT_RESULT message when load balancing initialization is complete. The response should come back very quickly. The GMD_INIT_RESULT message contains one field: a STR string field containing the subject name for which initialization has completed. The polling RTclient can use functions such as TipcSrvMsgSearchType to get the response. Both guaranteed message delivery and load balancing currently use the same accounting mechanism. Use TipcSrvSubjectLbInit, however, to initialize load balancing accounting for a subject. When the RTserver connected to the publishing RTclient routes a message for load balancing, it keeps track of which RTclients should receive the message. If this RTserver is not already tracking load balancing for subject when the first published message arrives from RTclient, the RTserver must start a monitoring-like watch mechanism between the RTservers of all subscribing RTclients. This increases the scalability of load balancing because only the relevant RTservers dynamically exchange load balancing information. The load balancing accounting continues until no messages have been published with GMD or load balancing for an interval specified by the Gmd_Publish_Timeout option. The default for Gmd_Publish_Timeout is 300 seconds.

The starting of this load balancing accounting is very fast but not instantaneous. TipcSrvSubjectLbInit is intended to be used by advanced publish-subscribe applications that wish to ensure that RTserver is primed and immediately ready for load balancing publish-subscribe routing.

TipcSrvSubjectLbInit is usually only used at initialization, but it can be called at any time by advanced applications which wish to initialize load balancing publishing accounting. The use of TipcSrvSubjectLbInit is strictly optional for load balancing publishing; if it is not called the load balancing setup occurs when the first GMD or load balancing message is published to *subject*.

If RTserver is already tracking load balancing for *subject*, RTserver immediately returns a GMD_INIT_RESULT message.

Caution TipcSrvSubjectLbInit sends a GMD_INIT_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

See Also TipcSrvSubjectLbInit, TipcSrvMsgSend, TipcSrvMsgWrite

Examples This example uses TipcSrvSubjectLbInit to initialize load balancing accounting in RTserver for the /system/thermal subject and then waits for the result message:

```
T_IPC_MT mt;
T_IPC_MSG result_msg;

/* initialize load balancing accounting in RTserver */
if (!TipcSrvSubjectLbInit("/system/thermal")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_GMD_INIT_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait a short time for the GMD init result */
result_msg = TipcSrvMsgSearchType(10.0, mt);
if (result_msg == NULL) {
    return; /* error */
}

/* must set current field first */
if (!TipcMsgSetCurrent(result_msg, 0)) {
    return; /* error */
}

/* at this point, we could read the first field of the result message to verify the subject name */
```


TipcSrvSubjectSetSubscribe

Name	TipcSrvSubjectSetSubscribe — start or stop subscribing to a subject
Synopsis	<pre>T_BOOL TipcSrvSubjectSetSubscribe(<i>subject</i>, <i>subscribe_status</i>) T_STR <i>subject</i>; T_BOOL <i>subscribe_status</i>;</pre>
Arguments	<i>subject</i> — name of the subject to set subscribe status for <i>subscribe_status</i> — subscribe status
Return Values	TRUE if the subscribe status of the subject was successfully set, FALSE otherwise.
Diagnostics	If TipcSrvSubjectSetSubscribe fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>subject</i> was null • T_ERR_VAL_INVALID — one of: — <i>subscribe_status</i> was not TRUE or FALSE, or — <i>subject</i> was the unique subject and <i>subscribe_status</i> was FALSE (RTclient cannot stop subscribing to its unique subject), or — <i>subject</i> was not a valid subject • T_ERR_VAL_SAME — <i>subscribe_status</i> is the same as the current subscribe status • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvSubjectSetSubscribe sets whether or not this RTclient is subscribing to a subject. The subscribe status controls whether or not the RTclient receives messages published (sent) to that subject. If the subscribe status of subject is TRUE, then the RTclient receives messages with <i>subject</i> as their destination that are sent to RTserver. To determine if an RTclient is subscribing to a subject, use TipcSrvSubjectGetSubscribe. TipcSrvSubjectSetSubscribe may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvSubjectSetSubscribe sets the subscribe status of the subject as if the RTclient were still connected to RTserver. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.

There are a number of standard subjects available to all RTclients. See `TpcSrvStdSubjectSetSubscribe` for more information on these standard subjects.

To determine if an RTclient is subscribing to a subject, with or without load balancing information, use `TpcSrvSubjectSetSubscribeLb`.

Caution `TpcSrvSubjectSetSubscribe` sends a `SUBJECT_SET_SUBSCRIBE` message to RTserver, but does not explicitly flush the message. See `TpcSrvGetAutoFlushSize` for more information on message buffering.

See Also `TpcMonClientSubscribeSetWatch`, `TpcMonSubjectSubscribeSetWatch`, `TpcSrvSubjectGetSubscribe`, `TpcSrvStdSubjectSetSubscribe`, `TpcSrvSubjectSetSubscribeLb`

Examples This example uses `TpcSrvSubjectSetSubscribe` to start subscribing to the subject `/system/thermal`:

```
if (!TpcSrvSubjectSetSubscribe("/system/thermal", TRUE)) {
    return; /* error */
}
```

TipcSrvSubjectSetSubscribeEx

Name	TipcSrvSubjectSetSubscribeEx — start or stop subscribing to a subject using extended options
Synopsis	<pre>T_BOOL TipcSrvSubjectSetSubscribeEx(subj_name, status, type, flags, type_specific_struct) T_STR subj_name; T_BOOL status; T_INT4 type; T_INT4 flags; T_PTR type_specific_struct;</pre>
Arguments	<i>subj_name</i> — name of the subject to set subscribe status for <i>status</i> — subscribe status (TRUE to subscribe or FALSE to unsubscribe) <i>type</i> — type of subscribe to use (T_SUBJ_NONE for a normal subscribe or T_SUBJ_CACHE to enable the SmartSockets Cache interface) <i>flags</i> — describe the attribute of <i>type</i> <i>type_specific_struct</i> — used to pass any additional information needed by a <i>type</i>
Return Values	TRUE if the subscribe status of the subject was successfully set, FALSE otherwise.
Diagnostics	If TipcSrvSubjectSetSubscribeEx fails, it returns FALSE and sets the global SmartSockets error number to one of: - T_ERR_DOESNT_EXIST — <i>type</i> was T_SUBJ_CACHE and caching options were not found, usually because caching was not initialized. See the <i>TIBCO SmartSockets Cache User's Guide</i> for more information. - any error number from TipcSrvSubjectSetSubscribe.
Description	TipcSrvSubjectSetSubscribeEx is used to subscribe to a subject using extended options. Currently the only option available is caching. If <i>type</i> is T_SUBJ_NONE, <i>flags</i> must be set to zero (0). If <i>type</i> is T_SUBJ_CACHE, <i>flags</i> must be one of: - T_SUBJ_FLAG_SC_INITIAL_LOAD — for force initial load - T_SUBJ_FLAG_SC_LATEST_MSG — to retrieve the last message from the cache or a new message if one arrives before the cache retrieval has completed If <i>type</i> is either T_SUBJ_NONE or T_SUBJ_CACHE, <i>type_specific_struct</i> must be set to T_NULL.
Caution	None

See Also None

Examples This example retrieves a message from SmartSockets Cache:

```
/* Retrieve message from Cache */
TutOut("Retrieving messages from Cache ...\n");

TipcSrvSubjectSetSubscribeEx(
    "/SCache/*",           /* requested subject */
    T_TRUE,                /* true to subscribe */
    T_SUBJ_CACHE,          /* type of subscribe */
    T_SUBJ_FLAG_SC_INITIAL_LOAD, /* attribute type to last value */
    T_NULL);               /* as using T_SUBJ_CACHE */
```

TipcSrvSubjectSetSubscribeLb

Name	TipcSrvSubjectSetSubscribeLb — start or stop subscribing to a subject, with or without load balancing
Synopsis	<pre>T_BOOL TipcSrvSubjectSetSubscribeLb(<i>subject</i>, <i>subscribe_status</i>, <i>lb_status</i>) T_STR <i>subject</i>; T_BOOL <i>subscribe_status</i>; T_BOOL <i>lb_status</i>;</pre>
Arguments	<i>subject</i> — name of the subject to set subscribe status for <i>subscribe_status</i> — subscribe status <i>lb_status</i> — load balancing status
Return Values	TRUE if the subscribe statuses of the subject were successfully set, FALSE otherwise.
Diagnostics	If TipcSrvSubjectSetSubscribeLb fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>subject</i> was null • T_ERR_VAL_INVALID — one of: — <i>subscribe_status</i> or <i>lb_status</i> were not TRUE or FALSE, or — <i>subject</i> was the unique subject and <i>subscribe_status</i> was FALSE (RTclient cannot stop subscribing to its unique subject), or — <i>subject</i> was not a valid subject • T_ERR_VAL_SAME — <i>subscribe_status</i> is the same as the current subscribe status • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created

Description TpcSrvSubjectGetSubscribeLb and TpcSrvSubjectSetSubscribeLb are variants of TpcSrvSubjectGetSubscribe and TpcSrvSubjectSetSubscribe that include load balancing control.

TpcSrvSubjectSetSubscribeLb sets whether or not this RTclient is subscribing to a subject, with or without load balancing. The subscribe status controls whether or not the RTclient receives messages published (sent) to that subject. If the subscribe status of *subject* is TRUE, then the RTclient receives messages with *subject* as their destination that are sent to RTserver. To determine if an RTclient is subscribing to a subject, including load balancing information, use TpcSrvSubjectGetSubscribeLb. If the load balancing status of *subject* is TRUE, then the RTclient is part of the load balancing calculation; otherwise the RTclient overrides load balancing and always receives the messages regardless of whether or not the message was published with load balancing.

TpcSrvSubjectSetSubscribeLb may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TpcSrvSubjectSetSubscribeLb sets the subscribe status of the subject as if the RTclient were still connected to RTserver. See TpcSrvCreate on automatic creation of connections and warm connections to RTserver.

Caution TpcSrvSubjectSetSubscribeLb sends a SUBJECT_SET_SUBSCRIBE message to RTserver, but does not explicitly flush the message. See TpcSrvGetAutoFlushSize for more information on message buffering.

See Also TpcMonClientSubscribeSetWatch, TpcMonSubjectSubscribeSetWatch, TpcSrvSubjectGetSubscribeLb, TpcSrvSubjectSetSubscribe

Examples This example uses TpcSrvSubjectSetSubscribeLb to start subscribing to the subject `/system/thermal` and override load balancing so that this RTclient receives all messages published to the `thermal` subject regardless of whether or not they were published with load balancing:

```
if (!TpcSrvSubjectSetSubscribeLb("/system/thermal", TRUE, FALSE))
{
    return; /* error */
}
```

TipcSrvSubjectTraverseSubscribe

Name	TipcSrvSubjectTraverseSubscribe — traverse the subjects that the RTclient is subscribing to
Synopsis	<pre>T_PTR TipcSrvSubjectTraverseSubscribe(<i>func</i>, <i>arg</i>) T_IPC_SRV_SUBJECT_TRAV_FUNC <i>func</i>; T_PTR <i>arg</i>;</pre>
Arguments	<i>func</i> — function to call once for each subject <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	The first non-null return value from <i>func</i> , or NULL if <i>func</i> always returns NULL.
Diagnostics	If TipcSrvSubjectTraverseSubscribe fails or traverses all the subjects an RTclient is subscribing to, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • any error number from TutHashTraverse
Description	TipcSrvSubjectTraverseSubscribe traverses all the subjects an RTclient is subscribing to (the subjects with a subscribe status of TRUE) and calls <i>func</i> once for each subject. If <i>func</i> returns a non-null value, then TipcSrvSubjectTraverseSubscribe returns with that value. TipcSrvSubjectTraverseSubscribe may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvSubjectTraverseSubscribe traverses the subjects as if the RTclient were still connected to RTserver. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvSubjectTraverseSubscribe returns NULL, it may be because there was an error or simply because <i>func</i> always returned NULL.
See Also	TipcMonClientSubscribePoll, TipcMonSubjectSubscribePoll; see the <i>TIBCO SmartSockets Utilities</i> for information on TutHashTraverse.

Examples This example prints the names of all subjects being subscribed to:

```
/* ===== */
/*..print_one_subject -- traverse one subject */
T_PTR print_one_subject(name, dummy, arg)
T_STR name;
T_STR dummy; /* not used */
T_PTR arg;
{
    TutOut("This process is subscribing to subject <%s>.\n", name);

    return NULL; /* continue traversal */
} /* print_one_subject */

/* ===== */
/*...code from calling function is below */

(void)TipcSrvSubjectTraverseSubscribe(print_one_subject, NULL);
```


TipcSrvTrafficGetBytesRecv8

Name TipcSrvTrafficGetBytesRecv8 — get the total number of bytes received on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

TipcSrvTrafficGetBytesRecv — get the total number of bytes received on a connection

Synopsis `T_BOOL TipcSrvTrafficGetBytesRecv8(bytes_recv_return)`
 `T_INT8 *bytes_recv_return;`

Arguments *bytes_recv_return* — returned number of bytes

Return Values TRUE if the number of bytes was successfully retrieved, FALSE otherwise.

Diagnostics If TipcSrvTrafficGetBytesRecv8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *bytes_recv_return* was null

Description TipcSrvTrafficGetBytesRecv8 retrieves the total number of bytes received on a connection.

See Also TipcSrvTrafficGetBytesSent8

Examples This example retrieves the number of bytes received on a connection:

```
T_INT8 bytes_recv;
.
.
.
if (!TipcSrvTrafficGetBytesRecv8(&bytes_recv)) {
    return; /* error */
}
```

TipcSrvTrafficGetBytesSent8

Name TipcSrvTrafficGetBytesSent8 — get the total number of sent received on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

TipcSrvTrafficGetBytesSent — get the total number of sent received on a connection

Synopsis `T_BOOL TipcSrvTrafficGetBytesSent8(bytes_sent_return)`
`T_INT8 *bytes_sent_return;`

Arguments *bytes_sent_return* — returned number of bytes

Return Values TRUE if the number of bytes was sucessfully retrieved, FALSE otherwise.

Diagnostics If TipcSrvTrafficGetBytesSent8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *bytes_sent_return* was null

Description TipcSrvTrafficGetBytesSent8 retrieves the total number of bytes sent on a connection.

See Also TipcSrvTrafficGetBytesRecv8

Examples This example retrieves the number of bytes sent on a connection:

```
T_INT8 bytes_sent;
.
.
.
if (!TipcSrvTrafficGetBytesSent8(&bytes_sent)) {
    return; /* error */
}
```

TipcSrvTrafficGetMsgsRecv8

Name TipcSrvTrafficGetMsgsRecv8 — get the total number of messages received on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

TipcSrvTrafficGetMsgsRecv — get the total number of messages received on a connection

Synopsis `T_BOOL TipcSrvTrafficGetMsgsRecv8(msgs_recv_return)`
 `T_INT8 *msgs_recv_return;`

Arguments *msgs_recv_return* — returned number of messages

Return Values TRUE if the number of messages was successfully retrieved, FALSE otherwise.

Diagnostics If TipcSrvTrafficGetMsgsRecv8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *msgs_recv_return* was null

Description TipcSrvTrafficGetMsgsRecv8 retrieves the total number of messages received on a connection.

See Also TipcSrvTrafficGetMsgsSent8

Examples This example retrieves the number of messages received on a connection:

```
T_INT8 msgs_recv;
.
.
.
if (!TipcSrvTrafficGetMsgsRecv8(&msgs_recv)) {
    return; /* error */
}
```

TipcSrvTrafficGetMsgsSent8

Name TipcSrvTrafficGetMsgsSent8 — get the total number of messages sent on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

TipcSrvTrafficGetMsgsSent — get the total number of messages sent on a connection

Synopsis `T_BOOL TipcSrvTrafficGetMsgsSent8(msgs_sent_return)`
 `T_INT8 *msgs_sent_return;`

Arguments *msgs_sent_return* — returned number of messages

Return Values TRUE if the number of messages was sucessfully retrieved, FALSE otherwise.

Diagnostics If TipcSrvTrafficGetMsgsSent8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *msgs_sent_return* was null

Description TipcSrvTrafficGetMsgsSent8 retrieves the total number of messages sent on a connection.

See Also TipcSrvTrafficGetMsgsRecv8

Examples This example retrieves the number of messages sent on a connection:

```
T_INT8 msgs_sent;
.
.
.
if (!TipcSrvTrafficGetMsgsSent8(&msgs_sent)) {
    return; /* error */
}
```

TipcSrvTraverseCbCreate

Name	TipcSrvTraverseCbCreate — create a server names traverse callback
Synopsis	<pre>T_CB TipcSrvTraverseCbCreate(<i>func</i>, <i>arg</i>) T_IPC_SRV_TRAVERSE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvTraverseCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcSrvTraverseCbCreate creates a callback that is called when an RTclient traverses through the Server_Names list. This callback is called for each server name in the list until either: • a successful connection is established, • all server names have been traversed, or • the stop_traverse field has been set to T_TRUE This callback is invoked by TipcSrvCreate, and is called before an attempt to connect to the RTserver currently being traversed. You can install the callback before or after connecting to the RTserver. See the <i>TIBCO SmartSockets Utilities</i> reference for more information on callbacks. See Server Names Traverse Callbacks for information on the data structure for this callback. When the callback function is called, the second function argument contains the callback data (C type T_IPC_SRV_TRAVERSE_CB_DATA), which includes the fields stop_traverse and server_name. The stop_traverse field, when set to T_TRUE, causes the process to stop traversing the Server_Names list and causes TipcSrvCreate to fail. The server_name field contains the current server name that is being traversed in the list. When an error occurs, this callback is called after the server reconnect delay occurs. Set the Server_Reconnect_Delay option to a low value if you want this callback to perform the initial delay.

Caution The T_ENTRY declaration specifier is required in the definition of all callback functions and their prototypes.

See Also TipcSrvTraverseCbLookup; see the *TIBCO SmartSockets Utilities* for information on TutCbDestroy.

Examples This example shows an application that creates a server names traverse callback used to print a message for each server name in the Server_Names list. It stops traversing the Server_Names list if a successful connection is made or connection attempts fail for the first three servers in the list:

```
#include <rtworks/ipc.h>

/* ===== */
/* ..my_server_traverse_cb - server name traversal callback */
void T_ENTRY my_server_traverse_cb
(
    T_IPC_CONN conn,
    T_IPC_SRV_TRAVERSE_CB_DATA data,
    T_CB_ARG arg
)
{
    T_INT4 *count = (T_INT4 *)arg;

    TutOut("Inside my_server_traverse_cb: server_name = %s\n",
        data->server_name);

    /* stop trying at 3 servers */
    if(*count >= 3) {
        TutWarning("Setting stop flag.\n");
        data->stop_traverse = T_TRUE;
    }
    *count += 1;
}

/* ===== */
/* ..main - main program */
int main(int argc, char **argv) {

    T_INT4 trav_count = 0;

    /* create a callback to be called when traversing the server names list */
    if (TipcSrvTraverseCbCreate(my_server_traverse_cb,
        (T_CB_ARG)&trav_count) == T_NULL) {
        return; /* error */;
    }

    TipcSrvCreate(T_IPC_SRV_CONN_FULL);

} /* main */
```

TipcSrvTraverseCbLookup

Name	TipcSrvTraverseCbLookup — look up a server names traverse callback
Synopsis	<pre>T_CB TipcSrvCreateCbLookup(<i>func</i>, <i>arg</i>) T_IPC_SRV_TRAVERSE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcSrvTraverseCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcSrvTraverseCbLookup looks up a server names traverse callback with the given function and argument. These callbacks are called just before an RTclient attempts to create a warm or full connection to an RTserver. See TipcSrvTraverseCbCreate for more information on server names traverse callbacks.
Caution	None
See Also	TipcSrvTraverseCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up a server names traverse callback: <pre>T_CB cb; cb = TipcSrvTraverseCbLookup(my_server_traversal_cb, NULL); if (cb == NULL) { return; /* error */ }</pre>

TipcSrvUnlock

Name	TipcSrvUnlock — release exclusive access to the connection to RTserver
Synopsis	T_BOOL TipcSrvUnlock()
Arguments	None
Return Values	TRUE if the connection to RTserver was successfully unlocked, FALSE otherwise.
Diagnostics	If TipcSrvUnlock fails, it returns FALSE and sets the global SmartSockets error number to: any error number from TutRwMutexUnlock
Description	TipcSrvUnlock unlocks all of the internal synchronization objects within the connection to RTserver.
Caution	Every call to TipcSrvLock must be balanced by a call to TipcSrvUnlock.
See Also	TipcConnUnlock, TipcSrvLock; see the <i>TIBCO SmartSockets Utilities</i> for information on TutRwMutexUnlock.
Examples	This example locks and unlocks the connection to RTserver:<pre>if (!TipcSrvLock()) { return; /* error */ }</pre> <i>/* At this point, several TipcSrv* functions could be called and there would be no risk of any other thread accessing the connection. The connection must be unlocked with TipcSrvUnlock when exclusive access is no longer needed. */</i> <pre>if (!TipcSrvUnlock()) { return; /* error */ }</pre>

TipcSrvWriteCbCreate

Name	TipcSrvWriteCbCreate — create a write callback in the connection to RTserver
Synopsis	<pre>T_CB TipcSrvWriteCbCreate(<i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_MT <i>mt</i>; T_IPC_CONN_WRITE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvWriteCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnWriteCbCreate
Description	TipcSrvWriteCbCreate creates a write callback in the connection to RTserver by calling TipcConnWriteCbCreate. These callbacks are called when a message is sent on the connection to RTserver. See TipcConnWriteCbCreate for more information on write callbacks. TipcSrvWriteCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvWriteCbCreate creates a write callback in the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnWriteCbCreate, TipcSrvWriteCbLookup

Examples This example creates a global write callback in the connection to RTserver that prints the name of the message type for each outgoing message:

```
/* ===== */
/*..my_srv_write_cb -- print type name of outgoing message */
void T_ENTRY my_srv_write_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_WRITE_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_IPC_MSG msg = data->msg;
    T_IPC_MT mt;
    T_STR name;

    if (!TipcMsgGetType(msg, &mt)) {
        return; /* error */
    }

    if (!TipcMtGetName(mt, &name)) {
        return; /* error */
    }

    /* print the message type name */
    TutOut("Outgoing message to RTserver with type %s.\n", name);
} /* my_srv_write_cb */

/* ===== */
/*...code from main program is below */

if (TipcSrvWriteCbCreate(NULL, /* NULL signifies global */
                        my_srv_write_cb,
                        NULL) == NULL) {
    return; /* error */
}
```

TipcSrvWriteCbLookup

Name	TipcSrvWriteCbLookup — look up a write callback in the connection to RTserver
Synopsis	<pre>T_CB TipcSrvWriteCbLookup(<i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_MT <i>mt</i>; T_IPC_CONN_WRITE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	Callback if successful, NULL otherwise.
Diagnostics	If TipcSrvWriteCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnWriteCbLookup
Description	TipcSrvWriteCbLookup looks up a write callback with the desired message type, function, and argument in the connection to RTserver by calling TipcConnWriteCbLookup. These callbacks are called when a message is sent on the connection to RTserver. See TipcConnWriteCbCreate for more information on connection write callbacks. TipcSrvWriteCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvWriteCbLookup looks up a write callback in the warm connection. See TipcSrvCreate on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnWriteCbLookup, TipcSrvWriteCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example looks up and destroys a NUMERIC_DATA write callback in the connection to RTserver:

```
T_CB cb;
T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

cb = TipcSrvWriteCbLookup(mt, my_numeric_write_cb, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}
```

Chapter 14 **TipcSrvConn***

This chapter covers the TipcSrvConn API functions that handle multiple connections. For information on the TipcSrv API functions that handle single server connections, see Chapter 13, TipcSrv*. This chapter covers these TipcSrvConn API functions:

- *TipcSrvConnBufferGetReadSize*
- *TipcSrvConnBufferGetWriteSize*
- *TipcSrvConnCheck*
- *TipcSrvConnClose*
- *TipcSrvConnCloseCbCreate*
- *TipcSrvConnCloseCbLookup*
- *TipcSrvConnCreate*
- *TipcSrvConnCreateNamed*
- *TipcSrvConnDefaultCbCreate*
- *TipcSrvConnDefaultCbLookup*
- *TipcSrvConnDestroy*
- *TipcSrvConnErrorCbCreate*
- *TipcSrvConnErrorCbLookup*
- *TipcSrvConnFlush*
- *TipcSrvConnGetArch*
- *TipcSrvConnGetAutoFlushSize*
- *TipcSrvConnGetCurrentConnectedLcn*
- *TipcSrvConnGetConnStatus*
- *TipcSrvConnGetGmdDirName*
- *TipcSrvConnGetGmdMaxSize*
- *TipcSrvConnGetGmdNumPending*
- *TipcSrvConnGetNode*
- *TipcSrvConnGetNumQueued*

- *TipcSrvConnGetPid*
- *TipcSrvConnGetProject*
- *TipcSrvConnGetServerNamesList*
- *TipcSrvConnGetSocket*
- *TipcSrvConnGetTimeout*
- *TipcSrvConnGetUniqueSubject*
- *TipcSrvConnGetUser*
- *TipcSrvConnGetXtSource*
- *TipcSrvConnGmdFileCreate*
- *TipcSrvConnGmdFileDelete*
- *TipcSrvConnGmdMsgDelete*
- *TipcSrvConnGmdMsgServerDelete*
- *TipcSrvConnGmdMsgStatus*
- *TipcSrvConnKeepAlive*
- *TipcSrvConnLock*
- *TipcSrvConnMainLoop*
- *TipcSrvConnMsgInsert*
- *TipcSrvConnMsgNext*
- *TipcSrvConnMsgProcess*
- *TipcSrvConnMsgSearch*
- *TipcSrvConnMsgSearchType*
- *TipcSrvConnMsgSend*
- *TipcSrvConnMsgSendRpc*
- *TipcSrvConnMsgWrite*
- *TipcSrvConnMsgWriteVa*
- *TipcSrvConnOpen*
- *TipcSrvConnOpenCbCreate*
- *TipcSrvConnOpenCbLookup*
- *TipcSrvConnPrint*
- *TipcSrvConnProcessCbCreate*

- *TipcSrvConnProcessCbLookup*
- *TipcSrvConnQueueCbCreate*
- *TipcSrvConnQueueCbLookup*
- *TipcSrvConnRead*
- *TipcSrvConnReadCbCreate*
- *TipcSrvConnReadCbLookup*
- *TipcSrvConnSetAutoFlushSize*
- *TipcSrvConnSetCredentials*
- *TipcSrvConnSetGmdDirName*
- *TipcSrvConnSetGmdMaxSize*
- *TipcSrvConnSetProject*
- *TipcSrvConnSetServerNames*
- *TipcSrvConnSetServerNamesList*
- *TipcSrvConnSetTimeout*
- *TipcSrvConnSetUsernamePassword*
- *TipcSrvConnStdSubjectSetSubscribe*
- *TipcSrvConnSubjectCbCreate*
- *TipcSrvConnSubjectCbDestroyAll*
- *TipcSrvConnSubjectCbLookup*
- *TipcSrvConnSubjectDefaultCbCreate*
- *TipcSrvConnSubjectDefaultCbLookup*
- *TipcSrvConnSubjectGetDefaultPrefix*
- *TipcSrvConnSubjectGetSubscribe*
- *TipcSrvConnSubjectGetSubscribeLb*
- *TipcSrvConnSubjectGetUnique*
- *TipcSrvConnSubjectGmdInit*
- *TipcSrvConnSubjectLbInit*
- *TipcSrvConnSubjectSetDefaultPrefix*
- *TipcSrvConnSubjectSetSubscribe*
- *TipcSrvConnSubjectSetSubscribeEx*

- *TpcSrvConnSubjectSetSubscribeLb*
- *TpcSrvConnSubjectSetUnique*
- *TpcSrvConnSubjectTraverseSubscribe*
- *TpcSrvConnTrafficGetBytesRecv8*
- *TpcSrvConnTrafficGetBytesSent8*
- *TpcSrvConnTrafficGetMsgsRecv8*
- *TpcSrvConnTrafficGetMsgsSent8*
- *TpcSrvConnUnlock*
- *TpcSrvConnWriteCbCreate*
- *TpcSrvConnWriteCbLookup*

TipcSrvConnBufferGetReadSize

Name	TipcSrvConnBufferGetReadSize — get the total number of bytes in a connection's read buffer
Synopsis	<pre>T_BOOL TipcSrvConnBufferGetReadSize(srv, read_size_return) T_IPC_SRV srv; T_INT4 *read_size_return;</pre>
Arguments	<i>srv</i> — connection handle to the RTserver <i>read_size_return</i> — returned number of bytes
Return Values	TRUE if the number of bytes was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcSrvConnBufferGetReadSize fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>srv</i> or <i>read_size_return</i> was null
Description	TipcSrvConnBufferGetReadSize retrieves the total number of bytes in a connection's read buffer.
See Also	TipcSrvConnBufferGetWriteSize
Examples	This example retrieves the number of bytes in a connection's read buffer: <pre>T_IPC_SRV srv; T_INT4 read_size; . . . if (!TipcSrvConnBufferGetReadSize(srv, &read_size)) { return; /* error */ }</pre>

TipcSrvConnBufferGetWriteSize

Name	TipcSrvConnBufferGetWriteSize — get the total number of bytes in a connection’s write buffer
Synopsis	<pre>T_BOOL TipcSrvConnBufferGetWriteSize(srv, write_size_return) T_IPC_SRV srv; T_INT4 *write_size_return;</pre>
Arguments	<i>srv</i> — connection handle to the RTserver <i>write_size_return</i> — returned number of bytes
Return Values	TRUE if the number of bytes was sucessfully retrieved, FALSE otherwise.
Diagnostics	If TipcSrvConnBufferGetWriteSize fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>srv</i> or <i>write_size_return</i> was null
Description	TipcSrvConnBufferGetWriteSize retrieves the total number of bytes in a connection’s write buffer. The returned number is usually the same as the total number of bytes actually sent on the connection, but sometimes the returned number is larger than the number of bytes sent when the write buffer is not frequently flushed.
See Also	TipcSrvConnBufferGetReadSize
Examples	This example retrieves the number of bytes in a connection’s write buffer: <pre>T_IPC_SRV srv; T_INT4 write_size; . . . if (!TipcSrvConnBufferGetWriteSize(srv, &write_size)) { return; /* error */ }</pre>

TipcSrvConnCheck

Name	TipcSrvConnCheck — check if data can be read from or written to the connection
Synopsis	<pre>T_BOOL TipcSrvConnCheck(srv, check_mode, timeout) T_IPC_SRV srv; T_IO_CHECK_MODE check_mode; T_REAL8 timeout;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>check_mode</i> — how to check RTserver availability <i>timeout</i> — maximum number of seconds to wait for availability
Return Values	TRUE if data can be read or written, FALSE if <i>timeout</i> reached or an error occurred.
Diagnostics	If TipcSrvConnCheck fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnCheck
Description	TipcSrvConnCheck checks the connection to RTserver by calling TipcConnCheck. The connection can be checked if data can be read (<i>check_mode</i> is T_IO_CHECK_READ) or if data can be written (<i>check_mode</i> is T_IO_CHECK_WRITE). See TipcConnCheck for more information on checking connections. TipcSrvConnCheck may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnCheck sleeps for <i>timeout</i> seconds and returns FALSE. See TipcSrvConnRead on automatic creation of connections and warm connections to RTserver.
Caution	TipcSrvConnCheck is a low-level function that is normally not used directly by developers. TipcConnCheck is normally called only by TipcSrvConnFlush and TipcSrvConnRead.
See Also	TipcConnCheck, TipcSrvConnRead

Example This example uses TipcSrvConnCheck to check if data is available within 3.4 seconds for reading from the connection to RTserver:

```
if (!TipcSrvConnCheck(srv, T_IO_CHECK_READ, 3.4)) {  
    return; /* error */  
}
```

TipcSrvConnClose

Name	TipcSrvConnClose — close a connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnClose(srv, conn_status) T_IPC_SRV srv; T_IPC_SRV_CONN_STATUS conn_status;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>conn_status</i> — whether to keep a warm connection to RTserver (use <code>T_IPC_SRV_CONN_WARM</code>) or fully close the connection to RTserver (use <code>T_IPC_SRV_CONN_NONE</code>)
Return Values	TRUE if the connection to RTserver was successfully closed, FALSE otherwise.
Diagnostics	If TipcSrvConnClose fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_NULL_PTR</code> — <i>srv</i> was null • <code>T_ERR_VAL_INVALID</code> — <i>destroy_status</i> was not <code>T_IPC_SRV_CONN_WARM</code> or <code>T_IPC_SRV_CONN_NONE</code> • <code>T_ERR_SRV_NOT_CONNECTED</code> — <i>destroy_status</i> was <code>T_IPC_SRV_CONN_WARM</code> but there was no full connection to RTserver, or <i>destroy_status</i> was <code>T_IPC_SRV_CONN_NONE</code> and there was no (full or warm) connection to RTserver
Description	TipcSrvConnClose closes the connection to RTserver. There are two ways to close the connection to RTserver: • A full close (<code>T_IPC_SRV_CONN_NONE</code>) removes all of RTclient's RTserver-related information. Once this has occurred, RTclient can continue as if it had never been connected at all to RTserver. • A warm close (<code>T_IPC_SRV_CONN_WARM</code>) keeps as much RTserver-related information as possible. Once this has occurred, RTclient can easily reconnect to RTserver in the future and resume where it left off. After a warm close, RTclient can even partially operate as if it still had a connection to RTserver. Messages can be sent, and they are buffered until a full connection to RTserver is opened again. See TipcSrvConnOpen on automatic creation of connections and warm connections to RTserver.

Destroy Status T_IPC_SRV_NONE

If *close_status* is T_IPC_SRV_NONE, then TpcSrvConnClose removes all local subject information, destroys the connection, and finally calls the server destroy callbacks.

Destroy Status T_IPC_SRV_CONN_WARM

If *close_status* is T_IPC_SRV_CONN_WARM, then TpcSrvConnClose first closes the socket in the connection to RTserver. (As soon as the socket is closed, RTserver closes its own connection to RTclient.) TpcSrvConnClose then removes all buffered incoming messages from the connection read buffer and all buffered outgoing messages from the connection write buffer. All messages in the connection message queue are left in place, however. TpcSrvConnClose next calls TpcSrvConnGmdResend to resend any messages for GMD; this primes the warm connection with GMD messages so that sequence number order is preserved. The server close callbacks are then called.

Caution An RTclient using the TCP/IP protocol to connect to RTserver may lose outgoing messages if the process terminates without calling TpcSrvConnClose. TCP/IP's SO_LINGER option, which preserves data, is ignored when closing a socket that has non-blocking I/O enabled. While data loss rarely occurs on UNIX and OpenVMS, it can happen very frequently on Windows. TpcSrvConnClose sets the block mode of the connection to TRUE before closing the connection's socket, which forces the operating system to deliver all flushed outgoing messages.

Messages that have been sent to RTserver but not flushed may be lost after a call to TpcSrvConnClose.

Use TpcSrvConnClose and T_IPC_SRV_NONE with caution from a connection callback such as a connection process callback. Because TpcSrvConnClose and T_IPC_SRV_NONE completely destroy the connection, the program crashes if control from the callback is returned to a function such as TpcSrvMainLoop.

See Also TpcSrvConnOpen, TpcSrvConnGetConnStatus

Examples This example opens a connection to RTserver, sends a message, and then fully closes the connection to RTserver with TipcSrvConnClose:

```
T_IPC_MT mt;

/* Create a connection to RTserver. */
if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

/* Send a message and make sure it gets flushed out. */
if (!TipcSrvConnMsgWrite(srv, "/system/admin", mt, TRUE,
                        T_IPC_FT_STR, "speed_limit",
                        T_IPC_FT_REAL8, 65.0,
                        NULL)) {
    return; /* error */
}

if (!TipcSrvConnFlush(srv)) {
    return; /* error */
}

/* Fully close the connection to RTserver. */
if (!TipcSrvConnClose(srv, T_IPC_SRV_CONN_NONE)) {
    return; /* error */
}
```

TipcSrvConnCloseCbCreate

Name	TipcSrvConnCloseCbCreate — create a server close callback
Synopsis	<pre>T_CB TipcSrvConnCloseCbCreate(srv, func, arg) T_IPC_SRV srv; T_IPC_SRV_DESTROY_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnCloseCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> or <i>srv</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcSrvConnCloseCbCreate creates a server close callback. These callbacks are called when an RTclient closes its connection to RTserver. Server close callbacks are useful for breaking any dependencies that RTclient has on being connected to RTserver. See the <i>TIBCO SmartSockets Utilities</i> reference for more information on callbacks. Server open callbacks are the opposite of server close callbacks. Server open callbacks can be used to perform processing when an RTclient opens a connection to RTserver. See TipcSrvConnOpenCbCreate for more information on server open callbacks. When the callback function is called, the second function argument contains the callback data (C type T_IPC_SRV_CLOSE_CB_DATA). The fields <i>old_conn_status</i> and <i>new_conn_status</i> (both have C type T_IPC_SRV_CONN_STATUS) in this callback data can be used to determine the previous and current status of the connection to RTserver. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution The first argument passed to *func* is the current connection to RTserver. This argument is null if there is currently no connection to RTserver.

A server close callback should not call any of the TipcSrvConn* functions that attempt to automatically connect to RTserver, as this would cause the server close callback to reconnect to RTserver.

The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.

See Also TipcSrvConnOpenCbCreate, TipcSrvConnCloseCbLookup; see the *TIBCO SmartSockets Utilities* for information on TutCbDestroy.

Examples This example shows how to create a close callback on a connection to RTserver:

```
/*..my_server_close_cb -- server close callback */
void T_ENTRY my_server_close_cb(
T_IPC_SRV srv;
T_IPC_SRV_CLOSE_CB_DATA data;
T_CB_ARG arg;
)
{
    /* any required processing here */

}    /* my_server_destroy_cb */

/* ===== */
/*...code from main program is below */

if (TipcSrvConnCloseCbCreate(srv, my_server_close_cb, NULL) ==
NULL) {
    return; /* error */
}
```

TipcSrvConnCloseCbLookup

Name	TipcSrvConnCloseCbLookup — look up a server close callback
Synopsis	<pre>T_CB TipcSrvConnCloseCbLookup(srv, func, arg) T_IPC_SRV srv; T_IPC_SRV_DESTROY_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnCloseCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> or <i>srv</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcSrvConnCloseCbLookup looks up a server close callback with the desired function and argument. These callbacks are called when RTclient closes its connection to RTserver. See TipcSrvConnCloseCbCreate for more information on server close callbacks.
Caution	None
See Also	TipcSrvConnCloseCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up a server close callback: <pre>T_CB cb; cb = TipcSrvConnCloseCbLookup(srv, my_server_close_cb, NULL); if (cb == NULL) { return; /* error */ }</pre>

TipcSrvConnCreate

Name	TipcSrvConnCreate — create an RTclient connection handle to an RTserver
Synopsis	<pre>T_IPC_SRV TipcSrvConnCreate(unique_subject, project, server_names, default_subject_prefix) T_STR unique_subject; T_STR project; T_STR server_names; T_STR default_subject_prefix;</pre>
Arguments	<i>unique_subject</i> — unique subject to use for the connection, or NULL to use the value of the Unique_Subject option <i>project</i> — project to use for the connection, or NULL to use the value of the Project option <i>server_names</i> — comma delimited list of logical connection names to use for the connection, or NULL to use the value of the Server_Names option <i>default_subject_prefix</i> — default subject prefix to use for the connection, or NULL to use the value of the Default_Subject_Prefix option
Return Values	The new connection handle if successful, NULL otherwise.
Diagnostics	If TipcSrvConnCreate fails, it returns NULL and sets the global SmartSockets error number to: • T_ERR_VAL_INVALID — one or more of the arguments were invalid
Description	TipcSrvConnCreate creates an RTclient connection handle to an RTserver. The connection is not opened until TipcSrvConnOpen is called or until a function that requires a connection is called, such as TipcSrvConnMsgSend, and the Server_Auto_Connect option is set to TRUE.
Caution	None
See Also	TipcSrvConnDestroy
Examples	This example creates an RTclient/RTserver connection handle: <pre>T_IPC_SRV srv; srv = TipcSrvConnCreate("/trader", NULL, "tcp:moe, tcp:larry", NULL);</pre>

TipcSrvConnCreateNamed

Name	TipcSrvConnCreateNamed — creates a named RTclient connection handle to an RTserver
Synopsis	<pre>T_IPC_SRV TipcSrvConnCreateNamed(<i>name</i>) T_STR <i>name</i>;</pre>
Arguments	<i>name</i> — string representing the name of the connection
Return Values	The new connection handle is returned.
Diagnostics	None
Description	TipcSrvConnCreateNamed creates a named RTclient connection handle to an RTserver. The name is associated with named options, which are set by the SETNOPT command or by setting an option that was looked up by TutOptionNamedLookup. The connection is not opened until TipcSrvConnOpen is called or until a function that requires a connection is called, such as TipcSrvConnMsgSend, and the Server_Auto_Connect option is set to TRUE.
Caution	None
See Also	TipcSrvConnDestroy
Examples	This example creates a named RTclient/RTserver connection handle: <pre>T_IPC_SRV srv; TutCommandParseStr("setnopt client Unique_subject /trader"); TutCommandParseStr("setnopt client Server_Names tcp:moe, tcp:larry"); srv = TipcSrvConnCreateNamed("client");</pre>

TipcSrvConnDefaultCbCreate

Name	TipcSrvConnDefaultCbCreate — create a default callback in the connection
Synopsis	<pre>T_CB TipcSrvConnDefaultCbCreate(<i>srv</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_IPC_CONN_DEFAULT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnDefaultCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnDefaultCbCreate
Description	TipcSrvConnDefaultCbCreate creates a default callback in the connection to RTserver by calling TipcConnDefaultCbCreate. These callbacks are called by TipcSrvConnMsgProcess when there are no process callbacks for the type of message being processed. See TipcConnDefaultCbCreate for more information on default callbacks. TipcSrvConnDefaultCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnDefaultCbCreate creates a default callback in the warm connection. See TipcSrvConnCreate for more information on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnDefaultCbCreate, TipcSrvConnMsgProcess, TipcSrvConnDefaultCbLookup

Examples This example creates a default callback which warns that an unexpected message has arrived from the connection to RTserver:

```
void T_ENTRY my_srv_default_cb(conn, data, arg)
T_IPC_CONN conn; /* connection to RTserver */
T_IPC_CONN_DEFAULT_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    if (!TipcMsgPrintError(data->msg)) {
        return; /* error */
    }
} /* my_srv_default_cb */

/* ===== */
/*...code from calling function is shown below */

/* create default callback */
if (TipcSrvConnDefaultCbCreate(srv, my_srv_default_cb, NULL) ==
NULL) {
    return; /* error */
}
```

TipcSrvConnDefaultCbLookup

Name	TipcSrvConnDefaultCbLookup — look up a default callback in the connection
Synopsis	<pre>T_CB TipcSrvConnDefaultCbLookup(<i>srv</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_IPC_CONN_DEFAULT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnDefaultCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnDefaultCbLookup
Description	TipcSrvConnDefaultCbLookup looks up a default callback with the desired connection, function, and argument in the connection to RTserver by calling TipcConnDefaultCbLookup. These callbacks are called by TipcSrvConnMsgProcess when there are no process callbacks for the type of message being processed. See TipcConnDefaultCbCreate for more information on default callbacks. TipcSrvConnDefaultCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnDefaultCbLookup looks up a default callback in the warm connection. See TipcSrvConnCreate for more information on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcSrvConnDefaultCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This looks up and destroys a default callback in the connection to RTserver:

```
T_CB cb;

cb = TipcSrvConnDefaultCbLookup(srv, my_srv_default_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
if (!TutCbDestroy(cb)) {
    return; /* error */
}
```


TipcSrvConnDestroy

Name	TipcSrvConnDestroy — free resources associated with the RTclient/RTserver connection handle
Synopsis	<pre>T_BOOL TipcSrvConnDestroy(srv) T_IPC_SRV srv;</pre>
Arguments	<i>srv</i> — connection handle to destroy
Return Values	TRUE if the connection handle was successfully destroyed, FALSE otherwise.
Diagnostics	If TipcSrvConnDestroy fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>srv</i> was null
Description	TipcSrvConnDestroy destroys a connection. All memory for the connection is freed, and the connection is closed.
Caution	None
See Also	TipcSrvConnCreate
Examples	This example destroys the connection handle: <pre>if (!TipcSrvConnDestroy(srv)) { return; /* error */ }</pre>

TipcSrvConnErrorCbCreate

Name	TipcSrvConnErrorCbCreate — create an error callback in the connection
Synopsis	<pre>T_CB TipcSrvConnErrorCbCreate(<i>srv</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_IPC_CONN_ERROR_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnErrorCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnErrorCbCreate
Description	TipcSrvConnErrorCbCreate creates an error callback in the connection to RTserver by calling TipcConnErrorCbCreate. These callbacks are called when there is an unrecoverable error on the connection to RTserver. See TipcConnErrorCbCreate for more information on error callbacks. TipcSrvConnErrorCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnErrorCbCreate creates an error callback in the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver. When an RTclient creates a connection to RTserver with TipcSrvConnCreate, an error callback is created in the connection with function TipcCbSrvError, argument null, and priority 0. TipcCbSrvError restarts RTserver, reconnects to RTserver, starts subscribing to the necessary subjects again, and starts watching the necessary subjects again. If an RTclient needs to take any additional recovery steps, such as checking for application-specific hardware failures, it can create an error callback with a lower priority (which is called after TipcCbSrvError).
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.

See Also `TipcCbSrvError`, `TipcConnErrorCbCreate`, `TipcSrvConnErrorCbLookup`; see the *TIBCO SmartSockets Utilities* for information on `TutCbDestroy`.

Examples This example creates an error callback for the connection to RTserver that is called before the error callback `TipcCbSrvError`:

```
/* ===== */
/*..my_srv_error_cb -- example server error callback func */
void T_ENTRY my_srv_error_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_ERROR_CB_DATA data;
T_CB_ARG arg;
{
    /* perform necessary error recovery here */
} /* my_srv_error_cb */

/*...fragment that uses above function is shown below */

T_CB cb;
T_CB_PRIORITY priority;

cb = TipcSrvConnErrorCbLookup(srv, TipcCbSrvError, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbGetPriority(cb, &priority)) {
    return; /* error */
}

cb = TipcSrvConnErrorCbCreate(srv, my_srv_error_cb, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbSetPriority(cb, priority + 1)) {
    return; /* error */
}
```

TipcSrvConnErrorCbLookup

Name	TipcSrvConnErrorCbLookup — look up an error callback in the connection
Synopsis	<pre>T_CB TipcSrvConnErrorCbLookup(<i>srv</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_IPC_CONN_ERROR_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnErrorCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnErrorCbLookup
Description	TipcSrvConnErrorCbLookup looks up an error callback with the desired function and argument in the connection to RTserver by calling TipcConnErrorCbLookup. These callbacks are called when there is an unrecoverable error on the connection to RTserver. See TipcConnErrorCbCreate for more information on error callbacks. TipcSrvConnErrorCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnErrorCbLookup looks up an error callback in the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcSrvConnErrorCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example looks up an error callback in the connection to RTserver:

```
T_CB cb;

cb = TipcSrvConnErrorCbLookup(srv, my_srv_error_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
```

TipcSrvConnFlush

Name	TipcSrvConnFlush — flush buffered outgoing messages on the connection
Synopsis	<pre>T_BOOL TipcSrvConnFlush(srv) T_IPC_SRV srv;</pre>
Arguments	<i>srv</i> — connection handle to RTserver
Return Values	TRUE if any data was successfully flushed to the connection handle to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnFlush fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnFlush
Description	TipcSrvConnFlush flushes buffered outgoing messages on the connection handle to RTserver by calling TipcConnFlush. See TipcConnFlush for more information on flushing outgoing messages. TipcSrvConnFlush may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnFlush returns FALSE immediately. See TipcSrvConnOpen on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnFlush, TipcSrvConnMsgSend
Examples	This example shows how to use TipcSrvConnFlush to flush a message to the connection handle to RTserver: <pre>if (!TipcSrvConnMsgSend(srv, msg, TRUE)) { return; /* error */ } if (!TipcSrvConnFlush(srv)) { return; /* error */ }</pre>

TipcSrvConnGetArch

Name	TipcSrvConnGetArch — get the architecture name of the connection handle to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnGetArch(srv, arch_return) T_IPC_SRV srv; T_STR *arch_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>arch_return</i> — storage for RTserver architecture name
Return Values	TRUE if the architecture name was successfully retrieved from the connection handle to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetArch fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>arch_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetArch
Description	TipcSrvConnGetArch gets the architecture name of the RTserver this RTclient is connected to by calling TipcConnGetArch. The RTserver architecture name can be useful for diagnostic purposes. The architecture name is the SmartSockets architecture, using the format: <i>Machine_OperatingSystem</i> where: <i>Machine</i> is the hardware name of the system. <i>OperatingSystem</i> is the operating system used. For example: sun4_solaris This value is stored in the environment variable RTARCH. There is some overlap between TipcSrvConnGetArch and TipcSrvMonServerGeneralPoll, but TipcSrvConnGetArch is easier to use if only the architecture name is needed instead of all RTserver general information.

TipcSrvConnGetArch may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetArch does not use the warm connection, because the warm connection does not have a peer process with an architecture name. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution If TipcSrvConnGetArch returns FALSE, it does not store a value in *arch_return*. The value stored in *arch_return* points directly into an internal data structure and should not be modified.

See Also TipcConnGetArch, TipcSrvMonServerGeneralPoll, TipcSrvConnGetNode, TipcSrvConnGetPid, TipcSrvConnGetUniqueSubject, TipcSrvConnGetUser

Examples This example prints the architecture name of the connected RTserver:

```
T_IPC_SRV srv;
T_STR arch_return;

if (!TipcSrvConnGetArch(srv, &arch_return)) {
    return; /* error */
}

TutOut("RTserver architecture name is <%s>.\n", arch_return);
```


TipcSrvConnGetAutoFlushSize

Name	TipcSrvConnGetAutoFlushSize — determine the auto flush size of the connection
Synopsis	<pre>T_BOOL TipcSrvConnGetAutoFlushSize(srv, auto_flush_size_return) T_IPC_SRV srv; T_INT4 *auto_flush_size_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>auto_flush_size_return</i> — storage for auto flush size
Return Values	TRUE if the auto flush size was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetAutoFlushSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>auto_flush_size_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetAutoFlushSize
Description	TipcSrvConnGetAutoFlushSize gets the auto flush size of the connection to RTserver by calling TipcConnGetAutoFlushSize. The auto flush size is used to control how much outgoing data (in bytes) is buffered before being automatically flushed to RTserver. The default for the auto flush size is 8192 bytes. See TipcConnGetAutoFlushSize for more information on auto flush sizes. TipcSrvConnGetAutoFlushSize may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetAutoFlushSize gets the auto flush size of the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	Unlike other connections, the auto flush size T_IPC_NO_AUTO_FLUSH_SIZE cannot be used with the connection to RTserver to disable all automatic flushing of outgoing messages. If TipcSrvConnGetAutoFlushSize returns FALSE, it does not store a value in <i>auto_flush_size_return</i>.
See Also	TipcConnGetAutoFlushSize, TipcSrvConnSetAutoFlushSize

Examples This example creates a connection to RTserver, then gets and prints the auto flush size of the connection to RTserver:

```
T_INT4 auto_flush_size;

if (!TpcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}

if (!TpcSrvConnGetAutoFlushSize(srv, &auto_flush_size)) {
    return; /* error */
}

TutOut("The connection to RTserver auto flush size is %d bytes.\n",
        auto_flush_size);
```

TipcSrvConnGetCurrentConnectedLcn

Name	TipcSrvConnGetCurrentConnectedLcn — get the logical connection name of the connected RTserver
Synopsis	<pre>T_BOOL TipcSrvConnGetCurrentConnectedLcn(srv, lcn_return) T_IPC_SRV srv; T_STR lcn_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>lcn_return</i> — storage for RTserver logical connection name
Return Values	TRUE if logical connection name was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcSrvConnGetCurrentConnectedLcn fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>srv</i> or <i>lcn_return</i> were null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver does not exist
Description	TipcSrvConnGetCurrentConnectedLcn gets the logical connection name of the RTserver to which this RTclient is connected. If the function cannot find the logical connection name, it stores the string Unknown in <i>lcn_return</i> .
Caution	There is no way to know ahead of time how much storage is needed for <i>lcn_return</i>. The value of <i>lcn_return</i> cannot be NULL. Storage for <i>lcn_return</i> must be managed by the caller. If TipcSrvConnGetCurrentConnectedLcn returns FALSE, it does not store a value in <i>lcn_return</i>.
See Also	TipcSrvGetCurrentConnectedLcn
Examples	This example prints the name of the logical connection name of RTserver: <pre>T_IPC_SRV srv; T_CHAR lcn[80]; if (!TipcSrvConnGetCurrentConnectedLcn(srv, lcn)) { return; /* error */ } TutOut("Current Logical Connection Name is <%s>\n", lcn);</pre>

TipcSrvConnGetConnStatus

Name TipcSrvConnGetConnStatus — get the status of the connection

Synopsis `T_BOOL TipcSrvConnGetConnStatus(srv, conn_status_return)
T_IPC_SRV srv;
T_IPC_SRV_CONN_STATUS *conn_status_return;`

Arguments *srv* — connection handle to RTserver
conn_status_return — storage for connection status

Return Values TRUE if connection status was successfully retrieved, FALSE otherwise.

Diagnostics If TipcSrvConnGetConnStatus fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *srv* or *conn_status_return* was null

Description TipcSrvConnGetConnStatus gets the status of the connection handle to RTserver. A connection status of T_IPC_SRV_CONN_FULL indicates that the RTclient does have a connection to RTserver. A connection status of T_IPC_SRV_CONN_WARM indicates that the RTclient has a warm connection to RTserver. A connection status of T_IPC_SRV_CONN_NONE indicates that the RTclient does not have a connection to RTserver. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver, and TipcSrvConnClose for more on destroying the connection to RTserver.

Caution If TipcSrvConnGetConnStatus returns FALSE, it does not store a value in *conn_status_return*.

See Also TipcSrvConnCreate, TipcSrvConnDestroy

Examples This example uses TipcSrvConnGetConnStatus to ensure that the RTclient has some kind of connection to RTserver:

```
T_IPC_SRV srv;  
T_IPC_SRV_CONN_STATUS conn_status_return;  
  
if (!TipcSrvConnGetConnStatus(srv, &conn_status_return)) {  
    return; /* error */  
}  
if (conn_status == T_IPC_SRV_CONN_NONE) {  
    TutOut("This process has no connection to RTserver.\n");  
    if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) {  
        return; /* error */  
    }  
}
```

TipcSrvConnGetGmdDirName

Name	TipcSrvConnGetGmdDirName — get the GMD area sub-directory name of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnGetGmdDirName(srv, gmd_dir_name_return) T_IPC_SRV srv; T_STR *gmd_dir_name_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>gmd_dir_name_return</i> — returned sub-directory name of the GMD area for the connection to RTserver
Return Values	TRUE if the GMD area sub-directory name was successfully returned for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetGmdDirName fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>srv</i> or <i>gmd_dir_name_return</i> was null
Description	TipcSrvConnGetGmdDirName returns the GMD area sub-directory name for the connection to RTserver.
Caution	None
See Also	TipcSrvConnSetGmdDirName
Examples	This example gets the GMD area sub-directory of the connection to RTserver: <pre>T_STR gmd_dir_name; if (!TipcSrvConnGetGmdDirName(srv, &gmd_dir_name)) { return; /* error */ }</pre>

TipcSrvConnGetGmdMaxSize

Name	TipcSrvConnGetGmdMaxSize — determine the GMD area maximum size of the connection
Synopsis	<pre>T_BOOL TipcSrvConnGetGmdMaxSize(srv, gmd_max_size_return) T_IPC_SRV srv; T_UINT4 *gmd_max_size_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>gmd_max_size_return</i> — storage for connection to RTserver GMD area maximum size
Return Values	TRUE if the GMD area maximum size was successfully retrieved from the connection handle to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetGmdMaxSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>gmd_max_size_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetGmdMaxSize
Description	TipcSrvConnGetGmdMaxSize gets the GMD area maximum size (in bytes) of the connection handle to RTserver by calling TipcConnGetGmdMaxSize. A GMD area holds guaranteed message delivery information for both incoming and outgoing messages. There are two types of guaranteed message delivery: file-based GMD and memory-only GMD. See TipcConnGmdFileCreate for more information on GMD areas. For file-based GMD this limit is the maximum file size, and for memory-only GMD this limit is the maximum amount of memory allowed. The default GMD area maximum size is 0, which means that no GMD area maximum size limit checking is performed. The connection GMD area maximum size and the connection delivery timeout can be used to constrain the amount of system resources used for GMD by space and by time. If the connection GMD area maximum size is exceeded, then no further messages can be sent with guaranteed delivery (TipcSrvConnMsgSend returns FALSE and sets the global SmartSockets error number to T_ERR_VAL_TOO_LARGE) until some unacknowledged previously sent messages are acknowledged.

TipcSrvConnGetGmdMaxSize may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetGmdMaxSize gets the GMD area maximum size of the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution If TipcSrvConnGetGmdMaxSize returns `FALSE`, it does not store a value in `gmd_max_size_return`.

See Also TipcConnGetGmdMaxSize, TipcSrvConnGmdFileCreate, TipcSrvConnSetGmdMaxSize

Examples This example prints the GMD area maximum size of the connection handle to RTserver:

```
T_IPC_SRV srv;
T_INT4 gmd_max_size_return;

if (!TipcSrvConnGetGmdMaxSize(srv, &gmd_max_size_return)) {
    return; /* error */
}

TutOut("Connection to RTserver GMD max size is %d.\n",
gmd_max_size);
```

TipcSrvConnGetGmdNumPending

Name	TipcSrvConnGetGmdNumPending — get the number of outgoing GMD messages still pending on the connection
Synopsis	<pre>T_BOOL TipcSrvConnGetGmdNumPending(srv, gmd_num_pending_return) T_IPC_SRV srv; T_INT4 *gmd_num_pending_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>gmd_num_pending_return</i> — storage for pending message count
Return Values	TRUE if the number of pending GMD messages was successfully retrieved from the connection handle to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetGmdNumPending fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>gmd_num_pending_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetGmdNumPending
Description	TipcSrvConnGetGmdNumPending gets the number of GMD messages published to RTserver that have not been acknowledged or explicitly deleted from the GMD area. Messages are deleted from the GMD area when a GMD_ACK message is received and when a GMD_FAILURE message is processed. TipcSrvConnGetGmdNumPending may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetGmdNumPending gets the number of pending GMD messages from the warm connection. See TipcSrvConnOpen for more information on creating connections and warm connections.
Caution	If TipcSrvConnGetGmdNumPending returns FALSE, it does not store a value in <i>gmd_num_pending_return</i> .
See Also	TipcConnGetGmdNumPending

Examples This example loops processing messages until there are no more outgoing GMD messages pending on the connection to RTserver:

```
T_IPC_SRV srv;
T_INT4 gmd_num_pending_return;
do {
    if (!TipcSrvConnMainLoop(srv, 1.0)) {
        return; /* error */
    }
    if (!TipcSrvConnGetGmdNumPending(srv, &gmd_num_pending_return))
{
    return; /* error */
}
} while (gmd_num_pending_return > 0);
```

TipcSrvConnGetNode

Name	TipcSrvConnGetNode — get the node name of the connected RTserver
Synopsis	<pre>T_BOOL TipcSrvConnGetNode(srv, node_return) T_IPC_SRV srv; T_STR *node_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>node_return</i> — storage for the RTserver node name
Return Values	TRUE if the node name was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetNode fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>node_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetNode
Description	TipcSrvConnGetNode gets the node name of the RTserver this RTclient is connected to by calling TipcConnGetNode. The RTserver node name can be useful for diagnostic purposes. There is some overlap between TipcSrvConnGetNode and TipcSrvMonServerGeneralPoll, but TipcSrvConnGetNode is easier to use if only the node name is needed instead of all RTserver general information. TipcSrvConnGetNode may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetNode does not use the warm connection, because the warm connection does not have a peer process with a node name. See TipcSrvConnOpen for more information on creating connections and warm connections to RTserver.
Caution	If TipcSrvConnGetNode returns FALSE, it does not store a value in <i>node_return</i>. The value stored in <i>node_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcConnGetNode, TipcSrvMonServerGeneralPoll, TipcSrvConnGetArch, TipcSrvConnGetPid, TipcSrvConnGetUniqueSubject, TipcSrvConnGetUser

Examples This example prints the node name of the connected RTserver:

```
T_IPC_SRV srv;  
T_STR node_return;  
  
if (!TipcSrvConnGetNode(srv, &node_return)) {  
    return; /* error */  
}  
TutOut("RTserver node name is <%s>.\n", node);
```

TipcSrvConnGetNumQueued

Name	TipcSrvConnGetNumQueued — get number of queued messages from the connection
Synopsis	<pre>T_BOOL TipcSrvConnGetNumQueued(srv, num_queued_return) T_IPC_SRV srv; T_INT4 *num_queued_return;</pre>
Arguments	<i>srv</i> — connection handle <i>num_queued_return</i> — storage for number of queued messages
Return Values	TRUE if number of queued messages was successfully retrieved from the connection handle to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetNumQueued fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>num_queued_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetNumQueued
Description	TipcSrvConnGetNumQueued gets the number of queued messages from the connection handle to RTserver by calling TipcConnGetNumQueued. A connection has a priority queue of incoming messages. See TipcConnGetNumQueued for more information on connection message queues. TipcSrvConnGetNumQueued may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetNumQueued gets the number of queued messages from the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvConnGetNumQueued returns FALSE, it does not store a value in <i>num_queued_return</i> .
See Also	TipcConnGetNumQueued, TipcSrvConnMsgNext, TipcSrvConnMainLoop, TipcSrvConnRead

Examples This example prints the number of queued messages, then reads from the connection to RTserver until at least three messages are available:

```
T_IPC_SRV srv;
T_INT4 num_queued_return;
if (!TipcSrvConnGetNumQueued(srv, &num_queued_return)) {
    return; /* error */
}
TutOut("The connection to RTserver has %d queued messages.\n",
       num_queued_return);

while (num_queued_return < 3) {
    if (!TipcSrvConnRead(srv, T_TIMEOUT_FOREVER)) {
        return; /* error */
    }
    if (!TipcSrvConnGetNumQueued(srv, &num_queued_return)) {
        return; /* error */
    }
}
```

TipcSrvConnGetPid

Name	TipcSrvConnGetPid — get the process ID of the connected RTserver
Synopsis	<pre>T_BOOL TipcSrvConnGetPid(srv, pid_return) T_IPC_SRV srv; T_INT4 *pid_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>pid_return</i> — storage for the RTserver process ID
Return Values	TRUE if the process ID was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetPid fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>pid_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetPid
Description	TipcSrvConnGetPid gets the operating system process ID of the RTserver this RTclient is connected to by calling TipcConnGetPid. The RTserver process ID can be useful for diagnostic purposes. There is some overlap between TipcSrvConnGetPid and TipcSrvMonServerGeneralPoll, but TipcSrvConnGetPid is easier to use if only the process ID is needed instead of all RTserver general information. TipcSrvConnGetPid may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetPid does not use the warm connection, because the warm connection does not have a peer process with a process ID. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvConnGetPid returns FALSE, it does not store a value in <i>pid_return</i> .
See Also	TipcConnGetPid, TipcSrvMonServerGeneralPoll, TipcSrvConnGetArch, TipcSrvConnGetNode, TipcSrvConnGetUniqueSubject, TipcSrvConnGetUser

Examples This example prints the process ID of the connected RTserver:

```
T_IPC_SRV srv;  
T_INT4 pid_return;  
if (!TipcSrvConnGetPid(srv, &pid_return)) {  
    return; /* error */  
}  
  
TutOut("RTserver ID is <%d>.\n", pid_return);
```

TipcSrvConnGetProject

Name TipcSrvConnGetProject — get the project of the connection to RTserver

Synopsis

```
T_BOOL TipcSrvConnGetProject(srv, project_return)
T_IPC_SRV srv;
T_STR *project_return;
```

Arguments *srv* — connection to RTserver
project_return — returned project

Return Values TRUE if the project was successfully retrieved for the connection to RTserver, FALSE otherwise.

Diagnostics If TipcSrvConnGetProject fails, it returns FALSE and sets the global SmartSockets error number to:

- T_ERR_NULL_PTR — *srv* or *project_return* was null

Description TipcSrvConnGetProject gets the project of the connection to RTserver.

Caution The storage pointed to by *project_return* should not be modified or freed. The storage is thread safe, but may be overwritten by subsequent calls to SmartSockets APIs within the same thread. If you wish to preserve the value, you should make a copy of the value.

See Also TipcSrvConnSetProject

Examples This example gets the RTclient’s project:

```
T_STR project;

if (!TipcSrvConnGetProject(srv, &project)) {
    return; /* error */
}
```


TipcSrvConnGetServerNamesList

Name	TipcSrvConnGetServerNamesList — get the server names of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnGetServerNamesList(srv, server_names_list_return) T_IPC_SRV srv; T_STR_LIST *server_names_list_return;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>server_names_list_return</i> — storage for the string list of server names
Return Values	TRUE if the server names were successfully retrieved for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetServerNamesList fails, it returns FALSE and sets the global SmartSockets error number to: T_ERR_NULL_PTR — <i>srv</i> or <i>server_names_list_return</i> was null
Description	TipcSrvConnGetServerNamesList gets the server names of the connection to RTserver.
Caution	The storage pointed to by <i>server_names_list_return</i> should not be modified or freed. The storage is thread safe, but may be overwritten by subsequent calls to SmartSockets APIs within the same thread. If you wish to preserve the value, you should make a copy of the value.
See Also	TipcSrvConnSetServerNamesList, TipcSrvConnSetServerNames
Examples	This example gets the RTclient's server names: <pre>T_STR_LIST server_names; if (!TipcSrvConnGetServerNamesList(srv, &server_names)) { return; /* error */ }</pre>

TipcSrvConnGetSocket

Name	TipcSrvConnGetSocket — determine the socket of the connection
Synopsis	<pre>T_BOOL TipcSrvConnGetSocket(<i>srv</i>, <i>socket_return</i>) T_IPC_SRV <i>srv</i>; T_INT4 *<i>socket_return</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>socket_return</i> — storage for connection socket
Return Values	TRUE if socket was successfully retrieved from the connection handle to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetSocket fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>socket_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetSocket
Description	TipcSrvConnGetSocket gets the socket file descriptor of the connection to RTserver by calling TipcConnGetSocket. A socket is an operating system device which provides a communication link to another process. All data sent to and received from the connection is transmitted on this socket. TipcSrvConnGetSocket may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetSocket does not use the warm connection, because the warm connection does not have a valid socket. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvConnGetSocket returns FALSE, it does not store a value in <i>socket_return</i> .
See Also	TipcConnGetSocket, TipcSrvConnGetXtSource

Examples This example gets the connection handle to RTserver and prints the socket of the connection:

```
T_IPC_SRV srv;
T_INT4 socket_return;
if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}
if (!TipcSrvConnGetSocket(srv, &socket_return)) {
    return; /* error */
}
TutOut("Socket of connection to RTserver is <%d>.\n",
socket_return);
```

TipcSrvConnGetTimeout

Name	TipcSrvConnGetTimeout — get a timeout property from the connection
Synopsis	<pre>T_BOOL TipcSrvConnGetTimeout(<i>srv</i>, <i>timeout</i>, <i>value_return</i>) T_IPC_SRV <i>srv</i>; T_IPC_TIMEOUT <i>timeout</i>; T_REAL8 *<i>value_return</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTServer <i>timeout</i> — which timeout property to get <i>value_return</i> — storage for timeout value
Return Values	TRUE if the timeout property was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetTimeout fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_VAL_INVALID — <i>timeout</i> was not a valid T_IPC_TIMEOUT • T_ERR_NULL_PTR — <i>srv</i> or <i>value_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetTimeout
Description	TipcSrvConnGetTimeout gets a timeout property from the connection to RTserver by calling TipcConnGetTimeout. The timeout properties control how often (in seconds) the connection to RTserver is checked for possible network failures. See TipcConnGetTimeout for more information on connection timeouts. The timeout properties of the connection to RTserver are also available in the options Server_Read_Timeout, Server_Write_Timeout, Server_Keep_Alive_Timeout, and Server_Delivery_Timeout. For the connection to RTserver, the default read timeout is 30.0 seconds, the default write timeout is 30.0 seconds, the default keep alive timeout is 15.0 seconds, and the default delivery timeout is 30.0 seconds. TipcSrvConnGetTimeout may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetTimeout gets a timeout property from the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvConnGetTimeout returns FALSE, it does not store a value in <i>value_return</i> .

See Also TipcConnGetTimeout, TipcSrvConnSetTimeout

Examples This example gets the read timeout from the connection to RTserver with TipcSrvConnGetTimeout and prints it:

```
T_IPC_SRV srv;
T_REAL8 timeout;

if (!TipcSrvConnGetTimeout(srv, T_IPC_TIMEOUT_READ, &timeout)) {
    return; /* error */
}

TutOut("The connection to RTserver's read timeout is %s
seconds.\n",
       TutRealToStr(timeout));
```

TipcSrvConnGetUniqueSubject

Name	TipcSrvConnGetUniqueSubject — get the unique subject of the connected RTserver
Synopsis	<pre>T_BOOL TipcSrvConnGetUniqueSubject(srv, unique_subject_return) T_IPC_SRV srv; T_STR *unique_subject_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>unique_subject_return</i> — storage for RTserver unique subject
Return Values	TRUE if the unique subject was successfully retrieved from the connection handle to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetUniqueSubject fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>unique_subject_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetUniqueSubject
Description	TipcSrvConnGetUniqueSubject gets the unique subject of the RTserver this RTclient is connected to by calling TipcConnGetUniqueSubject. The RTserver unique subject can be useful for diagnostic purposes. The unique subject, which is used to uniquely identify a process, is stored in the option Unique_Subject. There is some overlap between TipcSrvConnGetUniqueSubject and TipcSrvMonServerGeneralPoll, but TipcSrvConnGetUniqueSubject is easier to use if only the unique subject is needed instead of all RTserver general information. TipcSrvConnGetUniqueSubject may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetUniqueSubject does not use the warm connection, because the warm connection does not have a peer process with a unique subject. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvConnGetUniqueSubject returns FALSE, it does not store a value in <i>unique_subject_return</i>. The value stored in <i>unique_subject_return</i> points directly into an internal data structure and should not be modified.

See Also TipcConnGetUniqueSubject, TipcSrvMonServerGeneralPoll,
 TipcSrvConnGetArch, TipcSrvConnGetNode, TipcSrvConnGetPid,
 TipcSrvConnGetUser

Examples This example retrieves and prints the unique subject of the connected RTserver:

```
T_IPC_SRV srv;  
T_STR unique_subject;  
  
if (!TipcSrvConnGetUniqueSubject(srv, &unique_subject)) {  
    return; /* error */  
}  
  
TutOut("RTserver unique subject is <%s>.\n",  
        unique_subject);
```

TipcSrvConnGetUser

Name	TipcSrvConnGetUser — get the user name of the connection handle to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnGetUser(srv, user_return) T_IPC_SRV srv; T_STR *user_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>user_return</i> — storage for RTserver user name
Return Values	TRUE if the user name was successfully retrieved from the connection handle to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetUser fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>user_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetUser
Description	TipcSrvConnGetUser gets the user name of the RTserver this RTclient is connected to by calling TipcConnGetUser. The RTserver username can be useful for diagnostic purposes. There is some overlap between TipcSrvConnGetUser and TipcSrvMonServerGeneralPoll, but TipcSrvConnGetUser is easier to use if only the user name is needed instead of all RTserver general information. TipcSrvConnGetUser may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetUser does not use the warm connection, because the warm connection does not have a peer process with a user name. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvConnGetUser returns FALSE, it does not store a value in <i>user_return</i>. The value stored in <i>user_return</i> points directly into an internal data structure and should not be modified.
See Also	TipcConnGetUser, TipcSrvMonServerGeneralPoll, TipcSrvConnGetArch, TipcSrvConnGetNode, TipcSrvConnGetPid, TipcSrvConnGetUniqueSubject

Examples This example retrieves and prints the user name of the connected RTserver:

```
T_IPC_SRV srv;  
T_STR user_return;  
  
if (!TipcSrvConnGetUser(srv, &user_return)) {  
    return; /* error */  
}  
TutOut("RTserver user name is <%s>.\n", user_return);
```

TipcSrvConnGetXtSource

Name	TipcSrvConnGetXtSource — get source suitable for XtAppAddInput from the connection
Synopsis	<pre>T_BOOL TipcSrvConnGetXtSource(srv, source_return) T_IPC_SRV srv; T_INT4 *source_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>source_return</i> — storage for source
Return Values	TRUE if source was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGetXtSource fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>source_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGetXtSource
Description	TipcSrvConnGetXtSource gets an XtAppAddInput-compatible source from the connection to RTserver by calling TipcConnGetXtSource. This source allows SmartSockets IPC to be mixed with Motif or any other Xt-based widget set. See TipcConnGetXtSource for more information on Xt-compatible sources. TipcSrvConnGetXtSource may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGetXtSource does not use the warm connection, because the warm connection does not have a valid source. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvConnGetXtSource returns FALSE, it does not store a value in <i>source_return</i> .
See Also	TipcConnGetXtSource, TipcSrvConnGetSocket

Examples This example creates a connection to RTserver, then gets and prints the Xt-compatible source of the connection:

```
T_IPC_SRV srv;
T_INT4 source;

if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}
if (!TipcSrvConnGetXtSource(srv, &source)) {
    return; /* error */
}
TutOut("Source of connection to RTserver is <%d>.\n", source);
```

TipcSrvConnGmdFileCreate

Name	TipcSrvConnGmdFileCreate — create guaranteed message delivery area on the connection
Synopsis	<pre>T_BOOL TipcSrvConnGmdFileCreate(srv) T_IPC_SRV srv;</pre>
Arguments	<i>srv</i> — connection handle to RTserver
Return Values	TRUE if the GMD area was successfully created for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGmdFileCreate fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGmdFileCreate
Description	TipcSrvConnGmdFileCreate creates a GMD area on the connection to RTserver by calling TipcConnGmdFileCreate. A GMD area holds guaranteed message delivery information for both incoming and outgoing messages. There are two types of guaranteed message delivery: file-based GMD and memory-only GMD. See TipcConnGmdFileCreate for more information on GMD areas. The full pathnames of the GMD files for the connection to RTserver are simpler than those for other connections. The full pathnames are built from the value from TipcGetGmdDir plus <i>rtserver</i>. For example, an RTclient process with the unique subject <i>unique_subject</i> would use the full pathname: UNIX /tmp/rtworks/gmd/unique_subject/rtserver.hsn OpenVMS dka300:[ssuser.gmd.unique_subject]rtserver.hsn Windows c:\temp\rtworks\gmd\unique_subject\rtserver.hsn Because the connection to RTserver has a fixed purpose, it can use shorter file names than other generic connections.

TipcSrvConnGmdFileCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnGmdFileCreate creates the GMD area for the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution TipcSrvConnOpen calls TipcSrvConnGmdResend to automatically resend old GMD messages, and TipcSrvConnGmdResend calls TipcSrvConnGmdFileCreate if old GMD files exist.

The most common cause of TipcSrvConnGmdFileCreate switching to memory-only GMD is not setting the option Unique_Subject to a value other than the default _Node_Pid. Each process that uses file-based GMD must properly configure Unique_Subject.

See Also TipcConnGmdFileCreate, TipcSrvConnGmdFileDelete

Examples This example calls TipcSrvConnGmdFileCreate to explicitly create the GMD area for the connection to RTserver, and then sends a NUMERIC_DATA message with guaranteed delivery to the test subject through RTserver:

```
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_MSG msg;

/* have to set the Unique_Subject option to use file-based GMD */
TutCommandParseStr("setopt unique_subject /example/sender");

/* Explicitly create GMD area to allow full control over error handling. This is slightly tricky since
TipcSrvConnGmdFileCreate may call TipcSrvConnCreate which calls TipcSrvConnGmdResend
which calls TipcSrvConnGmdFileCreate, so the first call can fail with error number
T_ERR_ALREADY_EXISTS. */
if (!TipcSrvConnGmdFileCreate(srv)
    && TutErrNumGet() != T_ERR_ALREADY_EXISTS) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (!TipcSrvConnMsgWrite(srv, "/system/test", mt, FALSE,
    T_IPC_PROP_DELIVERY_MODE, T_IPC_DELIVERY_ALL,
    T_IPC_FT_STR, "test_value",
    T_IPC_FT_REAL8, 10.0,
    NULL)) {
    return; /* error */
}
```

TipcSrvConnGmdFileDelete

Name	TipcSrvConnGmdFileDelete — delete guaranteed message delivery files for the connection
Synopsis	<pre>T_BOOL TipcSrvConnGmdFileDelete(srv) T_IPC_SRV srv;</pre>
Arguments	<i>srv</i> — connection handle to RTserver
Return Values	TRUE if the connection GMD files were successfully deleted from disk, FALSE otherwise.
Diagnostics	If TipcSrvConnGmdFileDelete fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • T_ERR_IN_USE — the connection GMD files are already open by this process and cannot be removed • T_ERR_C — a C error has occurred. Call the function TutErrNumGetC to obtain the related C-specific error value. • any error number from TipcGetGmdDir
Description	TipcSrvConnGmdFileDelete deletes from disk guaranteed message delivery (GMD) files for the connection to RTserver. Because TipcSrvConnOpen calls TipcSrvConnGmdResend (which opens any existing GMD files), TipcSrvConnGmdFileDelete does not call TipcConnGmdFileDelete but instead uses the same algorithm as TipcConnGmdFileDelete. TipcSrvConnGmdFileDelete is useful for processes that wish to completely delete any obsolete GMD information. See TipcConnGmdFileDelete for more information on deleting GMD files.
Caution	Old GMD files must be removed before a warm or full connection to RTserver is created. TipcSrvConnGmdFileDelete does not delete the GMD files if they are already open by this process and in use for the connection to RTserver.
See Also	TipcConnGmdFileDelete, TipcSrvConnGmdFileCreate

Examples This example calls `TipcSrvConnGmdFileDelete` to remove any existing GMD files for the connection to RTserver, creates a connection to RTserver, and then sends a `NUMERIC_DATA` message with guaranteed delivery to the test subject using RTserver:

```
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_MSG msg;

/* have to set the Unique_Subject option to use file-based GMD */
TutCommandParseStr("setopt unique_subject /example/sender");

/* Clear out old GMD files before connecting to RTserver. No special action is needed if you wish to
instead resend any previously sent messages with GMD (TipcSrvConnCreate automatically calls
TipcSrvConnGmdResend). */
if (!TipcSrvConnGmdFileDelete(srv)) {
    return; /* error */
}

if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (!TipcSrvConnMsgWrite(srv, "/system/test", mt, FALSE,
                        T_IPC_PROP_DELIVERY_MODE, T_IPC_DELIVERY_ALL,
                        T_IPC_FT_STR, "test_value",
                        T_IPC_FT_REAL8, 10.0,
                        NULL)) {
    return; /* error */
}
```

TipcSrvConnGmdMsgDelete

Name	TipcSrvConnGmdMsgDelete — delete a message from the GMD area after a GMD failure
Synopsis	<pre>T_BOOL TipcSrvConnGmdMsgDelete(srv, msg) T_IPC_SRV srv; T_IPC_MSG msg;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>msg</i> — message to delete
Return Values	TRUE if the guaranteed message was successfully deleted from the GMD area of the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnGmdMsgDelete fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — the message was not previously sent with GMD • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnGmdMsgDelete
Description	TipcSrvConnGmdMsgDelete deletes a message after a GMD failure on the connection to RTserver by calling TipcConnGmdMsgDelete. TipcSrvConnGmdMsgDelete is intended to be used from a GMD_FAILURE connection process callback to terminate GMD for the message. See TipcConnGmdMsgDelete for more information on deleting messages when GMD failures occur. When a GMD_FAILURE message is processed by a sender RTclient, one of two scenarios is possible: • the sender RTclient does not want to resend the message, and thus uses TipcSrvConnGmdMsgDelete and TipcSrvConnGmdMsgServerDelete to terminate GMD for the message • the sender RTclient does want to resend the message, and thus takes some kind of user-defined action to correct the problem (possibly using TipcSrvConnGmdMsgStatus to poll RTserver for GMD status) and then resends the message

The standard callback `TipcCbConnProcessGmdFailure` uses `TipcConnGmdMsgDelete` to delete failed messages. Use `TipcSrvConnGmdMsgDelete` instead of `TipcConnGmdMsgDelete` in user-defined callbacks for processing `GMD_FAILURE` messages on the connection to RTserver. See `TipcCbConnProcessGmdFailure` and `TipcCbSrvProcessGmdFailure` for more information on processing `GMD_FAILURE` messages.

`TipcSrvConnGmdMsgDelete` may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, `TipcSrvConnGmdMsgDelete` deletes the message from the GMD area for the warm connection. See `TipcSrvConnOpen` for more information on automatic creation of connections and warm connections to RTserver.

Caution Use `TipcSrvConnGmdMsgDelete` only from a `GMD_FAILURE` process callback.

See Also `TipcCbConnProcessGmdFailure`, `TipcCbSrvProcessGmdFailure`, `TipcConnGmdMsgDelete`, `TipcSrvConnGmdMsgStatus`

Examples This example creates a connection process callback for `GMD_FAILURE` messages which silently deletes the message that failed (this callback is essentially a silent version of `TipcCbConnProcessGmdFailure`). The existing callback `TipcCbConnProcessGmdFailure` is destroyed so it does not interfere with the new callback:

```
void T_ENTRY my_gmd_failure_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_IPC_MSG failed_msg;
    T_IPC_SRV srv = arg;

    /* go to the first field of the GMD_FAILURE message */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* read the first field of the GMD_FAILURE message, which is */
    /* the message where GMD failed */
    if (!TipcMsgNextMsg(data->msg, &failed_msg))
        return; /* error */
    }

    /* terminate the sender's knowledge about the message for GMD */
    if (!TipcSrvConnGmdMsgDelete(srv, failed_msg)) {
        return; /* error */
    }
} /* my_gmd_failure_cb */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;
T_CB cb;

mt = TipcMtLookupByNum(T_MT_GMD_FAILURE);
if (mt == NULL) {
    return; /* error */
}

/* create connection process callback for GMD_FAILURE messages */
if (TipcSrvConnProcessCbCreate(srv, mt, my_gmd_failure_cb, srv)
    == NULL) {
    return; /* error */
}

/* destroy standard callback for GMD_FAILURE messages */
cb = TipcSrvConnProcessCbLookup(srv, mt,
                                TipcCbConnProcessGmdFailure,
                                NULL);

if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}

```

TipcSrvConnGmdMsgServerDelete

Name	TipcSrvConnGmdMsgServerDelete — delete a message in RTserver after a GMD failure
Synopsis	<pre>T_BOOL TipcSrvConnGmdMsgServerDelete(srv, msg) T_IPC_SRV srv; T_IPC_MSG msg;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>msg</i> — message for RTserver to delete
Return Values	TRUE if deletion of the message from RTserver was successfully initiated, FALSE otherwise.
Diagnostics	If TipcSrvConnGmdMsgServerDelete fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — the message was not previously sent with GMD • any error number from TipcSrvConnMsgWrite
Description	TipcSrvConnGmdMsgServerDelete informs RTserver to delete a message after a GMD failure on the connection to RTserver. TipcSrvConnGmdMsgServerDelete sends a GMD_DELETE message to RTserver. When RTserver routes a message for GMD, it keeps the message in memory so that it can easily and quickly resend the message to any receiving RTclient that disconnects and reconnects. TipcSrvConnGmdMsgServerDelete informs RTserver to terminate GMD for the message, which allows RTserver to reclaim the memory for the message. TipcSrvConnGmdMsgServerDelete is intended to be used from a GMD_FAILURE connection process callback to terminate GMD for the message. See TipcSrvConnGmdMsgDelete for more information on deleting messages when GMD failures occur. When a GMD_FAILURE message is processed by a sender RTclient, one of two scenarios is possible: • the sender RTclient does not want to resend the message, and thus uses TipcSrvConnGmdMsgDelete and TipcSrvConnGmdMsgServerDelete to terminate GMD for the message • the sender RTclient does want to resend the message, and thus takes some kind of user-defined action to correct the problem (possibly using

TpcSrvConnGmdMsgStatus to poll RTserver for GMD status) and then resends the message

The standard callback TpcCbSrvProcessGmdFailure uses TpcSrvConnGmdMsgServerDelete to delete failed messages. See TpcCbConnProcessGmdFailure and TpcCbSrvProcessGmdFailure for more information on processing GMD_FAILURE messages.

Caution Use TpcSrvConnGmdMsgServerDelete only from a GMD_FAILURE process callback.

TpcSrvConnGmdMsgServerDelete sends a GMD_DELETE message to RTserver, but does not explicitly flush the message. See TpcSrvConnGetAutoFlushSize for more information on message buffering.

See Also TpcCbConnProcessGmdFailure, TpcCbSrvProcessGmdFailure, TpcConnGmdMsgDelete, TpcSrvConnGmdMsgStatus

Examples This example creates a connection process callback for GMD_FAILURE messages which takes corrective actions to recover from the GMD failure. If recovery succeeds, then the message is resent, otherwise the message is deleted in the sender and in RTserver. The existing callbacks TpcCbConnProcessGmdFailure and TpcCbSrvProcessGmdFailure are destroyed so they do not interfere with the new callback:

```
void T_ENTRY my_gmd_failure_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_IPC_SRV srv = arg;
    T_IPC_MSG failed_msg;
    T_STR failed_proc_name;
    T_INT4 err_num;
    T_REAL8 send_time;

    /* go to the first field of the GMD_FAILURE message */
    if (!TpcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* read the fields of the GMD_FAILURE message */
    if (!TpcMsgNextMsg(data->msg, &failed_msg)
        || !TpcMsgNextStr(data->msg, &failed_proc_name)
        || !TpcMsgNextInt4(data->msg, &err_num)
        || !TpcMsgNextReal8(data->msg, &send_time)) {
        return; /* error */
    }
}
```

```

/* at this point, a sender would typically take some */
/* application-specific corrective action before resending */
/* the message */
    if (take_corrective_actions(failed_proc_name, err_num,
send_time)) {
        /* resend the message with GMD */
        if (!TipcSrvConnMsgSend(srv, failed_msg, TRUE)) {
            return; /* error */
        }
    }
    else {
        /* delete the message in the sender (this process) */
        if (!TipcSrvConnGmdMsgDelete(srv, failed_msg)) {
            return; /* error */
        }

        /* delete the message in RTserver */
        if (!TipcSrvConnGmdMsgServerDelete(srv, failed_msg)) {
            return; /* error */
        }
    }
} /* my_gmd_failure_cb */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;
T_CB cb;

mt = TipcMtLookupByNum(T_MT_GMD_FAILURE);
if (mt == NULL) {
    return; /* error */
}

/* create connection process callback for GMD_FAILURE messages */
if (TipcSrvConnProcessCbCreate(srv, mt, my_gmd_failure_cb,
                               srv) == NULL) {
    return; /* error */
}

/* destroy standard callbacks for GMD_FAILURE messages */

cb = TipcSrvConnProcessCbLookup(srv, mt,
TipcCbConnProcessGmdFailure,
                               NULL);

if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}

```

```
cb = TipcSrvConnProcessCbLookup(srv, mt,  
TipcCbSrvProcessGmdFailure, NULL);  
if (cb == NULL) {  
    return; /* error */  
}  
  
if (!TutCbDestroy(cb)) {  
    return; /* error */  
}
```

TipcSrvConnGmdMsgStatus

Name	TipcSrvConnGmdMsgStatus — poll RTserver for GMD status of a message
Synopsis	<pre>T_BOOL TipcSrvConnGmdMsgStatus(<i>srv</i>, <i>msg</i>) T_IPC_SRV <i>srv</i>; T_IPC_MSG <i>msg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>msg</i> — message to poll RTserver about
Return Values	TRUE if the poll for GMD status was successfully initiated, FALSE otherwise.
Diagnostics	If TipcSrvConnGmdMsgStatus fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — the message was not previously sent with GMD • any error number from TipcSrvConnMsgWrite
Description	TipcSrvConnGmdMsgStatus polls for the GMD status of <i>msg</i> by sending a GMD_STATUS_CALL message to RTserver. RTserver responds by sending back a GMD_STATUS_RESULT message. The response should come back very quickly. The polling RTclient can use functions such as TipcSrvConnMsgSearchType to get the response. When RTserver routes a message for GMD, it keeps track of which RTclients should receive the message and the status of GMD to each of these RTclients. TipcSrvConnGmdMsgStatus is intended to be used from a GMD_FAILURE connection process callback to query GMD status for the message. The GMD_STATUS_RESULT message contains four fields: • an INT4 integer field containing the sequence number property of the message • a STR_ARRAY string array field containing the names of the RTclients that have acknowledged successful delivery of the message • a STR_ARRAY string array field containing the names of the RTclients where GMD has failed for the message • a STR_ARRAY string array field containing the names of the RTclients where delivery is still pending for the message

If RTserver does not have an in-memory GMD record of *msg*, RTserver silently ignores the GMD_STATUS_CALL message and no GMD_STATUS_RESULT message is returned. This indicates that one of these has happened:

- RTserver has fully delivered *msg*
- TpcSrvConnGmdMsgServerDelete was used to terminate GMD for the message
- RTserver never routed *msg*

When a GMD_FAILURE message is processed by a sender RTclient, one of two scenarios is possible:

- the sender RTclient does not want to resend the message, and thus uses TpcSrvConnGmdMsgDelete and TpcSrvConnGmdMsgServerDelete to terminate GMD for the message
- the sender RTclient does want to resend the message, and thus takes some kind of user-defined action to correct the problem (possibly using TpcSrvConnGmdMsgStatus to poll RTserver for GMD status) and then resends the message

See TpcCbConnProcessGmdFailure and TpcCbSrvProcessGmdFailure for more information on processing GMD_FAILURE messages.

Caution TpcSrvConnGmdMsgStatus is usually only used from a GMD_FAILURE process callback, but it can be called at any time by advanced applications which wish to poll for GMD status.

TpcSrvConnGmdMsgStatus sends a GMD_STATUS_CALL message to RTserver, but does not explicitly flush the message. See TpcSrvConnGetAutoFlushSize for more information on message buffering.

See Also TpcCbConnProcessGmdFailure, TpcCbSrvProcessGmdFailure, TpcConnGmdMsgDelete, TpcSrvConnGmdMsgServerDelete

Examples This example creates a connection process callback for GMD_FAILURE messages which uses TipcSrvConnGmdMsgStatus to poll RTserver for GMD status. (This example callback simply prints the GMD status information. A real callback would take some corrective actions to recover from the GMD failure.)

```
void T_ENTRY my_gmd_failure_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_IPC_MSG failed_msg;
    T_IPC_MT mt;
    T_IPC_MSG result_msg;
    T_INT4 seq_num;
    T_STR *success_client_names;
    T_INT4 num_success_client_names;
    T_STR *failure_client_names;
    T_INT4 num_failure_client_names;
    T_STR *pending_client_names;
    T_INT4 num_pending_client_names;
    T_INT4 counter; /* loop counter */
    T_IPC_SRV srv = arg;

    /* go to the first field of the GMD_FAILURE message */
    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    /* read the first field of the GMD_FAILURE message, which is */
    /* the message where GMD failed */
    if (!TipcMsgNextMsg(data->msg, &failed_msg)) {
        return; /* error */
    }

    /* poll RTserver for GMD status information */
    if (!TipcSrvConnGmdMsgStatus(srv, failed_msg)) {
        return; /* error */
    }

    mt = TipcMtLookupByNum(T_MT_GMD_STATUS_RESULT);
    if (mt == NULL) {
        return; /* error */
    }

    /* wait a short time for the GMD status result */
    result_msg = TipcSrvConnMsgSearchType(srv, 10.0, mt);
    if (result_msg == NULL) {
        return; /* error */
    }

    /* must set current field first */
    if (!TipcMsgSetCurrent(result_msg, 0)) {
        return; /* error */
    }
}
```

```

/* get the fields from the message */
if (!TipcMsgRead(result_msg,
                 T_IPC_FT_INT4, &seq_num,
                 T_IPC_FT_STR_ARRAY, &success_client_names,
                 &num_success_client_names,
                 T_IPC_FT_STR_ARRAY, &failure_client_names,
                 &num_failure_client_names,
                 T_IPC_FT_STR_ARRAY, &pending_client_names,
                 &num_pending_client_names,
                 NULL)) {
    return; /* error */
}

TutOut("Got GMD_STATUS poll result.\n");
TutOut("seq_num = %d\n", seq_num);
TutOut("success_client_names\n");
for (counter = 0; counter < num_success_client_names; counter++)
{
    TutOut(" %s\n", success_client_names[counter]);
}
TutOut("failure_client_names\n");
for (counter = 0; counter < num_failure_client_names; counter++)
{
    TutOut(" %s\n", failure_client_names[counter]);
}
TutOut("pending_client_names\n");
for (counter = 0; counter < num_pending_client_names; counter++)
{
    TutOut(" %s\n", pending_client_names[counter]);
}

if (!TipcMsgDestroy(result_msg)) {
    return; /* error */
}
} /* my_gmd_failure_cb */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;
T_CB cb;

mt = TipcMtLookupByNum(T_MT_GMD_FAILURE);
if (mt == NULL) {
    return; /* error */
}

```

```

/* create connection process callback for GMD_FAILURE messages */
cb = TipcSrvConnProcessCbCreate(srv, mt, my_gmd_failure_cb, srv);
if (cb == NULL) {
    return; /* error */
}

/* raise priority of callback so it is called before */
/* TipcCbSrvProcessGmdFailure, which uses */
/* TipcSrvConnGmdMsgServerDelete to tell RTserver to forget */
/* the message */
if (!TutCbSetPriority(cb, 2)) {
    return; /* error */
}

```

TipcSrvConnKeepAlive

Name	TipcSrvConnKeepAlive — check the health of the connection to RTserver.
Synopsis	<pre>T_BOOL TipcSrvConnKeepAlive(srv) T_IPC_SRV srv;</pre>
Arguments	<i>srv</i> — connection handle to RTserver
Return Values	TRUE if RTserver responded in time, FALSE otherwise.
Diagnostics	If TipcSrvConnKeepAlive fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnKeepAlive
Description	TipcSrvConnKeepAlive checks if the connection to RTserver is still alive by calling TipcConnKeepAlive. This check is called a keep alive. See TipcConnKeepAlive for more information on keep alives. TipcSrvConnKeepAlive may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnKeepAlive immediately returns FALSE. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnKeepAlive
Examples	This example creates a connection to RTserver, then performs a keep alive every 10 seconds to check the health of the connection to RTserver: <pre>if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) { return; /* error */ } for (;;) { TutSleep(10.0); if (!TipcSrvConnKeepAlive(srv)) { return; /* error */ } }</pre>

TipcSrvConnLock

Name	TipcSrvConnLock — acquire exclusive access to the connection
Synopsis	<pre>T_BOOL TipcSrvConnLock(srv) T_IPC_SRV srv;</pre>
Arguments	<i>srv</i> — connection handle to RTserver
Return Values	TRUE if the connection to RTserver was successfully locked, FALSE otherwise.
Diagnostics	If TipcSrvConnLock fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • any error number from TutRwMutexWriteLock
Description	TipcSrvConnLock locks all of the internal synchronization objects within the connection to RTserver. This insures that the calling thread in a multi-threaded process has exclusive access to the connection to RTserver. TipcSrvConnLock may be called by a thread recursively.
Caution	Every call to TipcSrvConnLock must be balanced by a call to TipcSrvConnUnlock.
See Also	TipcConnLock, TipcSrvConnUnlock
Examples	This example locks and unlocks the connection to RTserver: <pre>if (!TipcSrvConnLock(srv)) { return; /* error */ }</pre> <i>/* At this point, several TipcSrvConn* functions could be called and there would be no risk of any other thread accessing the connection. The connection must be unlocked with TipcSrvConnUnlock when exclusive access is no longer needed. */</i> <pre>if (!TipcSrvConnUnlock(srv)) { return; /* error */ }</pre>

TipcSrvConnMainLoop

Name	TipcSrvConnMainLoop — read and process messages on the connection
Synopsis	<pre>T_BOOL TipcSrvConnMainLoop(<i>srv</i>, <i>timeout</i>) T_IPC_SRV <i>srv</i>; T_REAL8 <i>timeout</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>timeout</i> — maximum number of seconds to wait for message to arrive
Return Values	TRUE if all pending messages were successfully processed on the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnMainLoop fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMainLoop
Description	TipcSrvConnMainLoop reads and processes messages on the connection to RTserver by calling TipcConnMainLoop. TipcSrvConnMainLoop is a convenience function that can be used in many situations instead of TipcSrvConnMsgNext, TipcSrvConnMsgProcess, and TipcMsgDestroy. See TipcConnMainLoop for more information on reading and processing messages. If an error occurs, TipcSrvConnMainLoop by default restarts RTserver and returns FALSE. To use TipcSrvConnMainLoop with a timeout of T_TIMEOUT_FOREVER, wrap an extra forever loop, either for (;) or while (1), around the call to TipcSrvConnMainLoop. TipcSrvConnMainLoop may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnMainLoop processes messages on the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnMainLoop, TipcMsgDestroy, TipcSrvConnMsgNext, TipcSrvConnMsgProcess

Examples This example creates a connection to RTserver, then reads and processes messages forever with TipcSrvConnMainLoop (note the use of TipcSrvConnMainLoop in a loop):

```
if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) {  
    return; /* error */  
}  
  
for (;;) {  
    if (!TipcSrvConnMainLoop(srv, T_TIMEOUT_FOREVER)) {  
        return; /* error */  
    }  
}
```

TipcSrvConnMsgInsert

Name	TipcSrvConnMsgInsert — insert message into the queue of the connection
Synopsis	<pre>T_BOOL TipcSrvConnMsgInsert(<i>srv</i>, <i>msg</i>, <i>pos</i>) T_IPC_SRV <i>srv</i>; T_IPC_MSG <i>msg</i>; T_INT4 <i>pos</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>msg</i> — message to insert <i>pos</i> — position to insert message
Return Values	TRUE if the message was successfully inserted into the queue in the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnMsgInsert fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — <i>pos</i> was negative or larger than the number of messages in the queue • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgInsert
Description	TipcSrvConnMsgInsert inserts a message into the message queue of the connection to RTserver by calling TipcConnMsgInsert. A connection has a priority queue of incoming messages. See TipcConnMsgInsert for more information on inserting a message into a connection message queue. TipcSrvConnMsgInsert may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnMsgInsert inserts the message into the message queue of the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	Do not call TipcMsgDestroy to destroy the message passed to TipcSrvConnMsgInsert, because TipcSrvConnMsgInsert takes responsibility for destroying the message.
See Also	TipcConnMsgInsert, TipcSrvConnMsgNext

Examples This example constructs a message and inserts it onto the front of the message queue of the connection to RTserver:

```
T_IPC_MT mt;
T_IPC_MSG msg;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "temperature", 98.6)) {
    return; /* error */
}

if (!TipcSrvConnMsgInsert(srv, msg, 0)) {
    return; /* error */
}

/* At this point, msg "belongs" to the connection to RTserver, */
/* and should not be destroyed by the caller */
```

TipcSrvConnMsgNext

Name	TipcSrvConnMsgNext — retrieve the next message from the connection
Synopsis	<pre>T_IPC_MSG TipcSrvConnMsgNext(<i>srv</i>, <i>timeout</i>) T_IPC_SRV <i>srv</i>; T_REAL8 <i>timeout</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>timeout</i> — maximum number of seconds to wait for message to arrive
Return Values	A message if successful, NULL otherwise.
Diagnostics	If TipcSrvConnMsgNext fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgNext
Description	TipcSrvConnMsgNext gets the next message from the connection to RTserver by calling TipcConnMsgNext. A connection has a priority queue of incoming messages. See TipcConnMsgNext for more information on getting messages from a connection’s message queue. TipcSrvConnMsgNext may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, and if a message is immediately available, TipcSrvConnMsgNext gets the next message from the warm connection. If no messages are queued in the warm connection, then TipcSrvConnMsgNext calls TutSleep with timeout, so that RTclient does not use an abnormal amount of CPU time. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution Call `TipcMsgDestroy` to destroy the message returned by `TipcSrvConnMsgNext` when it is no longer needed, unless the message is passed to another function which takes responsibility for destroying the message (such as `TipcSrvConnMsgInsert`).

RTclients that receive messages with `TipcSrvConnMsgNext` or `TipcSrvConnMsgSearch` but do not call `TipcSrvConnMsgProcess` on those message are not monitorable with the `TipcSrvMon*` API or `RTmon`. RTclient monitoring is implemented using connection process callbacks, which are all called by `TipcSrvConnMsgProcess`.

See Also `TipcConnMsgNext`, `TipcSrvConnMsgProcess`, `TipcSrvConnMainLoop`, `TipcSrvConnRead`

Examples This example waits for up to 10 seconds to get the next message from the connection to RTserver, processes the message, and finally destroys the message:

```
T_IPC_MSG msg;

msg = TipcSrvConnMsgNext(srv, 10.0);
if (msg == NULL) {
    return; /* error */
}

if (!TipcSrvConnMsgProcess(srv, msg)) {
    return; /* error */
}

if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcSrvConnMsgProcess

Name	TipcSrvConnMsgProcess — process a message in the connection
Synopsis	<pre>T_BOOL TipcSrvConnMsgProcess(<i>srv</i>, <i>msg</i>) T_IPC_SRV <i>srv</i>; T_IPC_MSG <i>msg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>msg</i> — message to process
Return Values	TRUE if message was successfully processed, FALSE otherwise.
Diagnostics	If TipcSrvConnMsgProcess fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> or <i>srv</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgProcess
Description	TipcSrvConnMsgProcess processes a message in the connection to RTserver by calling TipcConnMsgProcess. The process callbacks or default callbacks are executed. See TipcConnMsgProcess for more information on processing messages. TipcSrvConnMsgProcess may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnMsgProcess processes the message using the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	RTclients that receive messages with TipcSrvConnMsgNext or TipcSrvConnMsgSearch but do not call TipcSrvConnMsgProcess on those message are not monitorable with the TipcSrvMon* API or RTmon. RTclient monitoring is implemented using connection process callbacks, which are all called by TipcSrvConnMsgProcess.
See Also	TipcConnMsgProcess, TipcSrvConnDefaultCbCreate, TipcSrvConnMsgNext, TipcSrvConnMainLoop, TipcSrvConnProcessCbCreate

Examples This example gets the next message from the connection to RTserver and processes it with TipcSrvConnMsgProcess:

```
msg = TipcSrvConnMsgNext(srv, T_TIMEOUT_FOREVER);
if (msg == NULL) {
    return; /* error */
}

if (!TipcSrvConnMsgProcess(srv, msg)) {
    return; /* error */
}

if (!TipcMsgDestroy(msg)) {
    return; /* error */
}
```

TipcSrvConnMsgSearch

Name	TipcSrvConnMsgSearch — search the message queue of the connection for a specific message
Synopsis	<pre>T_IPC_MSG TipcSrvConnMsgSearch(<i>srv</i>, <i>timeout</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_REAL8 <i>timeout</i>; T_IPC_CONN_MSG_SEARCH_FUNC <i>func</i>; T_PTR <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>timeout</i> — maximum number of seconds to wait for desired message to arrive <i>func</i> — function to call once for each message <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	A message if successful, NULL otherwise.
Diagnostics	If TipcSrvConnMsgSearch fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> or <i>srv</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgSearch
Description	TipcSrvConnMsgSearch searches the message queue of the connection to RTserver for a specific message by calling TipcConnMsgSearch. A connection has a priority queue of incoming messages. See TipcConnMsgSearch for more information on searching a connection’s message queue. TipcSrvConnMsgSearch may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, and if any messages are in the warm connection message queue, TipcSrvConnMsgSearch searches the messages in the warm connection. If no messages are queued in the warm connection, then TipcSrvConnMsgSearch calls TutSleep with timeout, so that RTclient does not use an abnormal amount of CPU time. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution The message returned by TipcSrvConnMsgSearch should be destroyed when it is no longer needed by calling TipcMsgDestroy, unless the message is passed to another function which takes responsibility for destroying the message, such as TipcSrvConnMsgInsert.

RTclients that receive messages with TipcSrvConnMsgNext or TipcSrvConnMsgSearch but do not call TipcSrvConnMsgProcess on those message are not monitorable with the TipcSrvMon* API or RTmon. RTclient monitoring is implemented using connection process callbacks, which are all called by TipcSrvConnMsgProcess.

See Also TipcConnMsgSearch, TipcSrvConnMsgSearchType

Examples This example waits for up to 10 seconds for a message to arrive that was sent to the /system/_all subject:

```
/* ===== */
/*..search_for_dest -- search one message */
static T_ENTRY T_BOOL search_for_dest(conn, msg, arg)
T_IPC_CONN conn;
T_IPC_MSG msg;
T_PTR arg; /* really (T_STR) */
{
    T_STR desired_dest = arg;
    T_STR dest;

    if (!TipcMsgGetDest(msg, &dest)) {
        return; /* error */
    }
    return strcmp(dest, desired_dest) == 0;
} /* search_for_dest */

/*...fragment that uses above search function */

msg = TipcSrvConnMsgSearch(srv, 10.0, search_for_dest,
"/system/_all");
if (msg == NULL) {
    return; /* error */
}
```

TipcSrvConnMsgSearchType

Name	TipcSrvConnMsgSearchType — search the message queue of a connection for a message with a specific type
Synopsis	<pre>T_IPC_MSG TipcSrvConnMsgSearchType(<i>srv</i>, <i>timeout</i>, <i>mt</i>) T_IPC_SRV <i>srv</i>; T_REAL8 <i>timeout</i>; T_IPC_MT <i>mt</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>timeout</i> — maximum number of seconds to wait for desired message to arrive <i>mt</i> — desired message type
Return Values	A message if successful, NULL otherwise.
Diagnostics	If TipcSrvConnMsgSearchType fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>mt</i> or <i>srv</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgSearchType
Description	TipcSrvConnMsgSearchType is a convenience function that searches for a message with a specific type by calling TipcConnMsgSearchType. A connection has a priority queue of incoming messages. See TipcConnMsgSearchType for more information on searching a connection’s message queue. TipcSrvConnMsgSearchType may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, and if any messages are in the warm connection message queue, TipcSrvConnMsgSearchType searches the messages in the warm connection. If no messages are queued in the warm connection, then TipcSrvConnMsgSearchType calls TutSleep with timeout, so that RTclient does not use an abnormal amount of CPU time. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution The message returned by `TipcSrvConnMsgSearchType` should be destroyed when it is no longer needed by calling `TipcMsgDestroy`, unless the message is passed to another function which takes responsibility for destroying the message (such as `TipcSrvConnMsgInsert`).

See Also `TipcConnMsgSearchType`, `TipcSrvConnMsgSearch`

Examples This example waits indefinitely for the next CONTROL message to arrive from the connection to RTserver:

```
mt = TipcMtLookupByNum(T_MT_CONTROL);
if (mt == NULL) {
    return; /* error */
}

msg = TipcSrvConnMsgSearchType(srv, T_TIMEOUT_FOREVER, mt);
if (msg == NULL) {
    return; /* error */
}
```

TipcSrvConnMsgSend

Name	TipcSrvConnMsgSend — publish (send) a message through the connection
Synopsis	<pre>T_BOOL TipcSrvConnMsgSend(srv, msg, check_server_msg_send) T_IPC_SRV srv; T_IPC_MSG msg; T_BOOL check_server_msg_send;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>msg</i> — message to send <i>check_server_msg_send</i> — whether or not to check the option <code>Server_Msg_Send</code> first
Return Values	TRUE if <i>msg</i> was successfully sent, FALSE otherwise.
Diagnostics	If TipcSrvConnMsgSend fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>msg</i> or <i>srv</i> was null, or the destination of <i>msg</i> was null • T_ERR_MSG_INVALID — <i>msg</i> was not a valid message • T_ERR_VAL_INVALID — <i>check_server_msg_send</i> was not TRUE or FALSE, or the destination of <i>msg</i> is not a valid subject • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgSend
Description	TipcSrvConnMsgSend sends a message for publish-subscribe through the connection to RTserver by calling TipcConnMsgSend. If <i>check_server_msg_send</i> is TRUE and the option <code>Server_Msg_Send</code> is FALSE, then TipcSrvConnMsgSend does not send the message, but returns TRUE immediately. Otherwise TipcSrvConnMsgSend first temporarily sets the sender of the message (using the value of the option <code>Unique_Subject</code>), then temporarily prepends the value of the option <code>Default_Subject_Prefix</code> to the message sender and destination if they do not start with slash (/), and then calls TipcConnMsgSend. See TipcConnMsgSend for more information on sending messages through connections. For normal usage, <i>check_server_msg_send</i> should always be TRUE. This allows the user to globally turn off all normal outgoing messages by setting the option <code>Server_Msg_Send</code> to FALSE. This is useful for running two RTclients (such as two RTie processes), the primary one with <code>Server_Msg_Send</code> set to TRUE and the backup one with <code>Server_Msg_Send</code> set to FALSE.

If RTclient has a warm connection to RTserver, TipcSrvConnMsgSend buffers the message as if the RTclient were still connected to RTserver. See TipcSrvConnOpen for more information on warm connections to RTserver.

Caution None

See Also TipcConnMsgSend, TipcSrvConnMsgWrite

Examples This example creates a NUMERIC_DATA message, appends fields, sets the destination subject, sends the message with TipcSrvConnMsgSend, sets the destination to a second subject, and sends the message again:

```
T_IPC_MT mt;
T_IPC_MSG msg;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

msg = TipcMsgCreate(mt);
if (msg == NULL) {
    return; /* error */
}

if (!TipcMsgAppendStrReal8(msg, "speed_limit", 65.0)) {
    return; /* error */
}

if (!TipcMsgSetDest(msg, "/system/admin")) {
    return; /* error */
}

/* send the message */
if (!TipcSrvConnMsgSend(srv, msg, TRUE)) {
    return; /* error */
}

if (!TipcMsgSetDest(msg, "/system/test")) {
    return; /* error */
}

/* send the message again, but to a different subject */
if (!TipcSrvConnMsgSend(srv, msg, TRUE)) {
    return; /* error */
}
```

TipcSrvConnMsgSendRpc

Name	TipcSrvConnMsgSendRpc — make a remote procedure call (RPC) with messages on the connection
Synopsis	<pre>T_IPC_MSG TipcSrvConnMsgSendRpc(<i>srv</i>, <i>call_msg</i>, <i>timeout</i>) T_IPC_SRV <i>srv</i>; T_IPC_MSG <i>call_msg</i>; T_REAL8 <i>timeout</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>call_msg</i> — call message for RPC <i>timeout</i> — maximum number of seconds to wait for result message to arrive
Return Values	A result message if successful, NULL otherwise.
Diagnostics	If TipcSrvConnMsgSendRpc fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>call_msg</i> was null, or the destination of <i>call_msg</i> was null • T_ERR_MSG_INVALID — <i>call_msg</i> was not a valid message • T_ERR_VAL_INVALID — destination of <i>call_msg</i> is not a valid subject • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnMsgSendRpc
Description	TipcSrvConnMsgSendRpc makes a remote procedure call (RPC) with messages on the connection to RTserver by calling TipcConnMsgSendRpc. One message is sent as the RPC call from the caller, and another message is sent back as the RPC result to the caller. See TipcConnMsgSendRpc for more information on RPCs that use message passing. TipcSrvConnMsgSendRpc may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnMsgSendRpc returns null immediately. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution When the message returned by `TipcSrvConnMsgSendRpc` is no longer needed, destroy it by calling `TipcMsgDestroy`, unless the message is passed to another function which takes responsibility for destroying the message, such as `TipcSrvConnMsgInsert`.

The message type number of the result message must be one greater than the message type number of the call message.

`TipcSrvConnMsgSendRpc` does not work well if multiple RTclients receive the call message, and thus all send a result message back to the caller.

`TipcSrvConnMsgSendRpc` returns the first result message in this case.

See Also `TipcConnMsgSendRpc`

Examples This example queries the current time of another RTclient with a message-based connection RPC. It sends a call message and waits for up to 10 seconds for a result message to be sent back from the other end of the connection. Shown first is a utility fragment that must be used by both the caller and the callee:

```
#define USER_MT_TIME_QUERY_CALL 100
#define USER_MT_TIME_QUERY_RESULT 101 /* call num plus one */

/* utility function to create message types */
static void T_ENTRY create_time_query_mt()
{
    T_IPC_MT mt;

    mt = TipcMtCreate("time_query_call",
                     USER_MT_TIME_QUERY_CALL,
                     "");

    if (mt == NULL) {
        return; /* error */
    }

    mt = TipcMtCreate("time_query_result",
                     USER_MT_TIME_QUERY_RESULT,
                     "real8");

    if (mt == NULL) {
        return; /* error */
    }
} /* create_time_query_mt */
```

Shown next is the caller fragment:

```
/* query current time via RPC */
T_REAL8 T_ENTRY time_query_rpc(unique_subject)
T_STR unique_subject; /* unique subject of remote process */
{
    T_IPC_MT mt;
    T_IPC_MSG call_msg;
    T_IPC_MSG result_msg;
    T_REAL8 result;
```

```

    mt = TipcMtLookupByNum(USER_MT_TIME_QUERY_CALL);
    if (mt == NULL) {
        return; /* error */
    }

    call_msg = TipcMsgCreate(mt);
    if (call_msg == NULL) {
        return; /* error */
    }
    if (!TipcMsgSetDest(call_msg, unique_subject)) {
        return; /* error */
    }

    /* send/rcv RPC messages */
    result_msg = TipcSrvConnMsgSendRpc(srv, call_msg, 10.0);
    if (result_msg == NULL) {
        return; /* error */
    }

    /* process result without using TipcSrvConnMsgProcess */

    if (!TipcMsgNextReal8(result_msg, &result)) {
        return; /* error */
    }
    if (!TipcMsgDestroy(result_msg)) {
        return; /* error */
    }
    if (!TipcMsgDestroy(call_msg)) {
        return; /* error */
    }
    return result;
} /* time_query_rpc */

```

This is the callee (the end opposite of the caller) fragment:

```

/* callback to process RPC "call" message */

static void T_ENTRY process_call(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
    T_STR caller; /* unique subject of caller process */

    mt = TipcMtLookupByNum(USER_MT_TIME_QUERY_RESULT);
    if (mt == NULL) {
        return; /* error */
    }

    /* get the sender property of the message, which is the */
    /* unique subject of the caller process */
    if (!TipcMsgGetSender(data->msg, &caller)) {
        return; /* error */
    }
}

```

```

/* send a result message back to the caller */
    if (!TipcSrvConnMsgWrite(srv, caller, mt, FALSE,
                            T_IPC_FT_REAL8, TutGetCurrentTime(),
                            NULL)) {
        return; /* error */
    }
    if (!TipcSrvConnFlush(srv)) {
        return; /* error */
    }
} /* process_call */

/*...fragment that uses process_call */
T_IPC_MT mt;

mt = TipcMtLookupByNum(USER_MT_TIME_QUERY_CALL);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt, process_call, NULL) ==
    NULL) {
    return; /* error */
}
TipcSrvConnMainLoop(srv, T_TIMEOUT_FOREVER); /* process call messages */

```

TipcSrvConnMsgWrite

Name	TipcSrvConnMsgWrite — construct a message and send it through the connection
Synopsis	<pre>T_BOOL TipcSrvConnMsgWrite(srv, dest, mt, check_server_msg_send, ...) T_IPC_SRV srv; T_STR dest; T_IPC_MT mt; T_BOOL check_server_msg_send;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>dest</i> — destination subject for message <i>mt</i> — message type to use for message <i>check_server_msg_send</i> — whether or not to check the option <code>Server_Msg_Send</code> first additional arguments — null-terminated field type-value pairs
Return Values	TRUE if the message was successfully constructed and sent, FALSE otherwise.
Diagnostics	If <code>TipcSrvConnMsgWrite</code> fails, it returns FALSE and sets the global <code>SmartSockets</code> error number to one of: • <code>T_ERR_NULL_PTR</code> — <i>srv</i>, <i>dest</i>, or <i>mt</i> was null • <code>T_ERR_VAL_INVALID</code> — <i>check_server_msg_send</i> was not TRUE or FALSE, or <i>dest</i> was not a valid subject • any error number from <code>TipcSrvConnMsgWriteVa</code>
Description	<code>TipcSrvConnMsgWrite</code> is a convenience function that takes a null-terminated variable number of field type-value pairs. The field type part of the pair is <code>T_IPC_FT</code>, and the value part of the pair corresponds to the arguments for the related <code>TipcMsgAppend*</code> function (if the type part is <code>T_IPC_FT_REAL8</code>, then the value part is a REAL8 number) or <code>TipcMsgSet*</code> function (if the type part is <code>T_IPC_PROP_DELIVERY_MODE</code>, then the value part is a <code>T_IPC_DELIVERY_MODE</code> setting). Some field types, such as arrays, use two arguments for the value part.

TipcSrvConnMsgWrite performs these steps:

1. Create a message of type *mt*
2. Append the fields to the message
3. Set the message sender to the value of the option Unique_Subject
4. Set the message destination to *dest*
5. Send the message with TipcSrvConnMsgSend on connection *srv*
6. Destroy the message with TipcMsgDestroy

TipcSrvConnMsgWrite is a short wrapper function that calls TipcSrvConnMsgWriteVa.

If *check_server_msg_send* is TRUE and the option *Server_Msg_Send* is FALSE, then TipcSrvConnMsgWrite does not send the message, but returns TRUE immediately. For normal usage, *check_server_msg_send* should always be TRUE. This allows the user to globally turn off all normal outgoing messages by setting the option *Server_Msg_Send* to FALSE. This is useful for running two RTclients, the primary one with *Server_Msg_Send* set to TRUE and the backup one with *Server_Msg_Send* set to FALSE.

TipcSrvConnMsgWrite may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnMsgWrite buffers the message as if the RTclient were still connected to RTserver. See TipcSrvConnCreate for more information on automatic creation of connections and warm connections to RTserver.

Caution None

See Also TipcConnMsgWrite, TipcSrvConnMsgSend, TipcSrvConnMsgWriteVa

Examples This example constructs and sends a NUMERIC_DATA message with TipcSrvConnMsgWrite:

```
T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

if (!TipcSrvConnMsgWrite(srv, "/primary/control", mt, TRUE,
                        T_IPC_FT_STR, "voltage1",
                        T_IPC_FT_REAL8, 65.0,
                        T_IPC_FT_STR, "current1",
                        T_IPC_FT_REAL8, 10.335,
                        NULL)) {
    return; /* error */
}
```

TipcSrvConnMsgWriteVa

Name	TipcSrvConnMsgWriteVa — construct a message and send it through the connection (<i>va_list</i> version)
Synopsis	<pre>T_BOOL TipcSrvConnMsgWriteVa(srv, dest, mt, check_server_msg_send, var_arg_list) T_IPC_SRV srv; T_STR dest; T_IPC_MT mt; T_BOOL check_server_msg_send; va_list var_arg_list;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>dest</i> — message destination <i>mt</i> — message type <i>check_server_msg_send</i> — whether or not to check the option <code>Server_Msg_Send</code> first <i>var_arg_list</i> — argument list
Return Values	TRUE if the message was successfully constructed and sent, FALSE otherwise.
Diagnostics	If <code>TipcSrvConnMsgWriteVa</code> fails, it returns <code>FALSE</code> and sets the global <code>SmartSockets</code> error number to one of: <code>T_ERR_NULL_PTR</code> — <i>srv</i>, <i>dest</i>, or <i>mt</i> was null <code>T_ERR_VAL_INVALID</code> — <i>check_server_msg_send</i> was not TRUE or FALSE, or <i>dest</i> was not a valid subject - any error number from <code>TipcMsgWriteVa</code> - any error number from <code>TipcSrvConnMsgSend</code>
Description	<code>TipcSrvConnMsgWriteVa</code> is the non-<code>varargs</code> version of <code>TipcSrvConnMsgWrite</code> (<code>TipcSrvConnMsgWrite</code> actually calls <code>TipcSrvConnMsgWriteVa</code>). <code>TipcSrvConnMsgWriteVa</code> can be used to implement a <code>varargs</code> function. In the C language, a <code>varargs</code> function (one that takes a variable number of arguments) cannot call another <code>varargs</code> function with the original arguments. A <code>varargs</code> function usually calls a helper function that takes a fixed number of arguments, with the last argument usually of type <code>va_list</code>. <code>TipcSrvConnMsgWriteVa</code> is <code>TipcSrvConnMsgWrite</code>'s helper function.
Caution	None
See Also	<code>TipcSrvConnMsgWrite</code> , <code>TipcMsgWrite</code> , <code>TipcSrvConnMsgSend</code>

Examples This example shows how to write a varargs function in ANSI C that is similar to TipcSrvConnMsgWrite:

```
T_BOOL T_ENTRY my_srv_msg_write(T_IPC_SRV srv,
                                T_STR dest,
                                T_IPC_MT mt,
                                ...)
{
    va_list var_arg_list;
    T_BOOL status;

    va_start(var_arg_list, mt);
    status = TipcSrvConnMsgWriteVa(srv, dest, mt, TRUE,
var_arg_list);
    va_end(var_arg_list);
    return status;
} /* my_srv_msg_write */
```

TipcSrvConnOpen

Name	TipcSrvConnOpen — open a connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnOpen(srv, create_status) T_IPC_SRV srv; T_IPC_SRV_CONN_STATUS create_status;</pre>
Arguments	<i>srv</i> — the connection to open <i>create_status</i> — whether to create a warm connection (use <code>T_IPC_SRV_CONN_WARM</code>) or a full connection (use <code>T_IPC_SRV_CONN_FULL</code>)
Return Values	TRUE if the connection to RTserver was successfully opened, FALSE otherwise.
Diagnostics	If TipcSrvConnOpen fails, it returns FALSE and sets the global SmartSockets error number to one of: • <code>T_ERR_VAL_INVALID</code> — <i>create_status</i> was not <code>T_IPC_SRV_CONN_WARM</code> or <code>T_IPC_SRV_CONN_FULL</code> • <code>T_ERR_SRV_ALREADY_CONNECTED</code> — the connection is already open • <code>T_ERR_SRV_ACCESS_DENIED</code> — access was denied because another RTclient or RTserver has the same value for the unique subject • <code>T_ERR_SRV_NOT_CONNECTED</code> — a connection to an RTserver could not be created
Description	TipcSrvConnOpen opens the connection to RTserver. The RTclient becomes part of the project, can subscribe to subjects, can publish (send) messages to RTserver, and receive messages from RTserver. Each project is self-contained; messages cannot be sent between projects. When the connection to RTserver is created, the RTclient immediately starts subscribing to its unique subject. The RTclient can call TipcSrvConnSubjectSetSubscribe to start subscribing to other subjects. An RTclient can have a warm connection to RTserver, which is a subset of a full connection to RTserver. Most of the other TipcSrvConn functions, such as TipcSrvConnMsgSend, call TipcSrvConnOpen if <code>Server_Auto_Connect</code> is set to TRUE and the RTclient does not have a full connection to RTserver. See TipcSrvConnClose for more information on warm connections to RTserver.
Caution	None
See Also	TipcSrvConnClose, TipcSrvConnGetConnStatus

Examples This example sets the options Project and Server_Names, then connects to RTserver:

```
T_IPC_SRV srv;

srv = TipcSrvConnCreate(T_NULL, "demo", "tcp:mode", T_NULL);
if (!srv) {
    return; /* error */
}

if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}
```

TipcSrvConnOpenCbCreate

Name	TipcSrvConnOpenCbCreate — create a server open callback
Synopsis	<pre>T_CB TipcSrvConnOpenCbCreate(srv, func, arg) T_IPC_SRV srv; T_IPC_SRV_CREATE_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnOpenCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> or <i>srv</i> was null • T_ERR_ALREADY_EXISTS — a callback with <i>func</i> and <i>arg</i> already exists
Description	TipcSrvConnOpenCbCreate creates a server open callback. These callbacks are called when an RTclient creates a warm or full connection to RTserver. See the <i>TIBCO SmartSockets Utilities</i> reference for more information on callbacks. Server close callbacks are the opposite of server open callbacks. Server close callbacks can be used to perform processing when RTclient closes its connection to RTserver. See TipcSrvConnCloseCbCreate for more information on server close callbacks. When the callback function is called, the second function argument contains the callback data (C type T_IPC_SRV_OPEN_CB_DATA). The fields <i>old_conn_status</i> and <i>new_conn_status</i> (both have C type T_IPC_SRV_CONN_STATUS) in this callback data can be used to determine the previous and current status of the connection to RTserver. The field <i>old_conn_status</i> contains either the value T_IPC_SRV_CONN_NONE or the value T_IPC_SRV_CONN_WARM. The field <i>new_conn_status</i> contains the value T_IPC_SRV_CONN_WARM or the value T_IPC_SRV_CONN_FULL. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.

See Also `TipcSrvConnOpenCbLookup`, `TipcSrvConnCloseCbCreate`; see the *TIBCO SmartSockets Utilities* for information on `TutCbDestroy`

Examples This example shows an application that creates a server open callback which is used to print a message when a new server connection is made:

```
#include <rtworks/ipc.h>

/* ===== *
/*..my_server_open_cb -- server open callback */
void T_ENTRY my_server_open_cb(srv, data, arg)
T_IPC_SRV srv;
T_IPC_SRV_OPEN_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    TutOut("Server connection opened.\n");
} /* my_server_open_cb */

/* ===== *
/*..main -- main program */
int main(int argc, char **argv)
{
    T_IPC_SRV srv;

    /* create a connection handle to RTserver */
    if ((srv = TipcSrvConnCreate(NULL, NULL, NULL, NULL)) == NULL) {
        return; /* error */
    }

    /* create a callback to be called whenever we connect */
    /* to RTserver */
    if (TipcSrvConnOpenCbCreate(my_server_open_cb, NULL) == NULL) {
        return; /* error */
    }

} /* main */
```

TipcSrvConnOpenCbLookup

Name	TipcSrvConnOpenCbLookup — look up a server open callback
Synopsis	<pre>T_CB TipcSrvConnOpenCbLookup(srv, func, arg) T_IPC_SRV srv; T_IPC_SRV_CREATE_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnOpenCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>func</i> or <i>srv</i> was null • T_ERR_DOESNT_EXIST — no such callback exists
Description	TipcSrvConnOpenCbLookup looks up a server open callback with the desired function and argument. These callbacks are called when an RTclient opens a warm or full connection to RTserver. See TipcSrvConnOpenCbCreate for more information on server create callbacks.
Caution	None
See Also	TipcSrvConnOpenCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up a server open callback: <pre>T_CB cb; cb = TipcSrvConnOpenCbLookup(my_server_open_cb, NULL); if (cb == NULL) { return; /* error */ }</pre>

TipcSrvConnPrint

Name	TipcSrvConnPrint — print all information about the connection in portable, reproducible format
Synopsis	<pre>T_BOOL TipcSrvConnPrint(srv, func) T_IPC_SRV srv; T_OUT_FUNC func;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>func</i> — TutOut-style output function
Return Values	TRUE if the connection to RTserver was successfully printed, FALSE otherwise.
Diagnostics	If TipcSrvConnPrint fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnPrint prints all the information about the connection to RTserver. TipcSrvConnPrint may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnPrint prints the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	See the <i>TIBCO SmartSockets Utilities</i> for information on TutOut.
Examples	This example creates a connection to RTserver and then prints information about the connection: <pre>if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) { return; /* error */ } if (!TipcSrvConnPrint(srv, TutOut)) { return; /* error */ }</pre>

The output from the above fragment is similar to:

```

Connecting to project <rtworks> on <_node> RTserver.
Using local protocol.
Message from RTserver: Connection established.
Start subscribing to subject </_workstation1.tibco.com_8493>.
Printing connection named <client:local:workstation1.tibco.com:RTSERVER>.
logical_name = local:workstation1.tibco.com:RTSERVER.
socket_fd = 4.
read_buf: current = 0, size = 0, max = 65536
write_buf: current = 0, size = 0, max = 65536
num_queued = 0
int format = BIG_ENDIAN
real format = IEEE
char format = 437
unique subject = /_workstation1.tibco.com_8491
node:user:pid = workstation1.tibco.com:ssuser:8491
arch = sun4_solaris
block mode = false
auto flush size = 8192
read timeout = 30
write timeout = 30
keep alive timeout = 5
delivery timeout = 30
gmd file name = rtserver
auto gmd mode = true
num error callbacks = 1
no accept callbacks
num encode callbacks = 1
no decode callbacks
num global read callbacks = 1
no read callbacks
no global write callbacks
no write callbacks
no global process callbacks
num msg types with process callbacks = 16
  msg type <mon_client_msg_rcv_set_watch> has 1 callbacks.
  msg type <mon_client_buffer_poll_call> has 1 callbacks.
  msg type <control> has 1 callbacks.
  msg type <mon_client_buffer_set_watch> has 1 callbacks.
  msg type <mon_client_time_poll_call> has 1 callbacks.
  msg type <mon_client_time_set_watch> has 1 callbacks.
  msg type <mon_client_general_poll_call> has 1 callbacks.
  msg type <keep_alive_call> has 1 callbacks.
  msg type <gmd_failure> has 2 callbacks.
  msg type <gmd_nack> has 1 callbacks.
  msg type <mon_client_subject_poll_call> has 1 callbacks.
  msg type <mon_client_msg_traffic_poll_call> has 1 callbacks.
  msg type <mon_client_cb_poll_call> has 1 callbacks.
  msg type <mon_client_msg_type_poll_call> has 1 callbacks.
  msg type <mon_client_msg_send_set_watch> has 1 callbacks.
  msg type <mon_client_option_poll_call> has 1 callbacks.
no default process callbacks
no global queue callbacks
no queue callbacks

```

TipcSrvConnProcessCbCreate

Name	TipcSrvConnProcessCbCreate — create a process callback in the connection
Synopsis	<pre>T_CB TipcSrvConnProcessCbCreate(srv, mt, func, arg) T_IPC_SRV srv; T_IPC_MT mt; T_IPC_CONN_PROCESS_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnProcessCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnProcessCbCreate
Description	TipcSrvConnProcessCbCreate creates a process callback in the connection to RTserver by calling TipcConnProcessCbCreate. These callbacks are called by TipcSrvConnMsgProcess to perform the normal processing of a message. See TipcConnProcessCbCreate for more information on process callbacks. TipcSrvConnProcessCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnProcessCbCreate creates a process callback in the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnProcessCbCreate, TipcSrvConnProcessCbLookup

Examples This example creates a process callback for TIME messages in the connection to RTserver which prints the new time:

```
/*===== */
/*..my_time_process_cb -- process TIME messages */
void T_ENTRY my_time_process_cb(conn, data, arg)
T_IPC_CONN conn; /* connection to RTserver */
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_REAL8 new_time;

    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    if (!TipcMsgNextReal8(data->msg, &new_time)) {
        return; /* error */
    }

    TutOut("New time is %s\n", TutRealToStr(new_time));
} /* my_time_process_cb */

/*===== */
/*...code from calling function is below */

mt = TipcMtLookupByNum(T_MT_TIME);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt, my_time_process_cb, NULL)
    == NULL) {
    return; /* error */
}
```

TipcSrvConnProcessCbLookup

Name	TipcSrvConnProcessCbLookup — look up a process callback in the connection
Synopsis	<pre>T_CB TipcSrvConnProcessCbLookup(srv, mt, func, arg) T_IPC_SRV srv; T_IPC_MT mt; T_IPC_CONN_PROCESS_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnProcessCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnProcessCbLookup
Description	TipcSrvConnProcessCbLookup looks up the process callback with the desired message type, function, and argument in the connection to RTserver by calling TipcConnProcessCbLookup. These callbacks are called by TipcSrvConnMsgProcess to perform the normal processing of a message. See TipcConnProcessCbCreate for more information on connection process callbacks. TipcSrvConnProcessCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnProcessCbLookup looks up a process callback in the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
See Also	TipcConnProcessCbLookup, TipcSrvConnProcessCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example looks up a global process callback in the connection to RTserver:

```
T_CB cb;

cb = TipcSrvConnProcessCbLookup(srv, NULL,
my_global_srv_process_cb,
                                NULL);

if (cb == NULL) {
    return; /* error */
}
```

TipcSrvConnQueueCbCreate

Name	TipcSrvConnQueueCbCreate — create a queue callback in the connection
Synopsis	<pre>T_CB TipcSrvConnQueueCbCreate(<i>srv</i>, <i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_IPC_MT <i>mt</i>; T_IPC_CONN_QUEUE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnQueueCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnQueueCbCreate
Description	TipcSrvConnQueueCbCreate creates a queue callback in the connection to RTserver by calling TipcConnQueueCbCreate. These callbacks are called when a message is inserted into or deleted from the message queue of the connection to RTserver. See TipcConnQueueCbCreate for more information on queue callbacks. TipcSrvConnQueueCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnQueueCbCreate creates a queue callback in the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnQueueCbCreate, TipcSrvConnQueueCbLookup

Examples This example creates a global queue callback in the connection to RTserver which prints the number of queued messages:

```
void T_ENTRY my_srv_queue_cb(conn, data, arg)
T_IPC_CONN conn; /* connection to RTserver */
T_IPC_CONN_QUEUE_CB_DATA data;
T_CB_ARG arg;
{
    T_INT4 num_queued;
    T_IPC_SRV srv = arg;

    if (!TIPC_SrvConnGetNumQueued(srv, &num_queued)) {
        return; /* error */
    }

    /* print the queue size */
    TutOut("Number of queued messages from RTserver is now %d.\n",
           num_queued);
} /* my_srv_queue_cb */

/* ===== */
/*...code from main program is below */

if (TIPC_SrvConnQueueCbCreate(srv, NULL, /* NULL signifies global */
                             my_srv_queue_cb,
                             srv) == NULL) {
    return; /* error */
}
```


TipcSrvConnQueueCbLookup

Name	TipcSrvConnQueueCbLookup — look up a queue callback in the connection
Synopsis	<pre>T_CB TipcSrvConnQueueCbLookup(<i>srv</i>, <i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_IPC_MT <i>mt</i>; T_IPC_CONN_QUEUE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnQueueCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnQueueCbLookup
Description	TipcSrvConnQueueCbLookup looks up a queue callback with the desired message type, function, and argument in the connection to RTserver by calling TipcConnQueueCbLookup. These callbacks are called when a message is inserted into or deleted from the message queue of the connection to RTserver. See TipcConnQueueCbCreate for more information on connection queue callbacks. TipcSrvConnQueueCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnQueueCbLookup looks up a queue callback in the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnQueueCbCreate, TipcSrvConnQueueCbCreate, TipcConnQueueCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example looks up and destroys a NUMERIC_DATA queue callback in the connection to RTserver:

```
T_CB cb;
T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
    return; /* error */
}

cb = TipcSrvConnQueueCbLookup(srv, mt, my_numeric_queue_cb, NULL);
if (cb == NULL) {
    return; /* error */
}

if (!TutCbDestroy(cb)) {
    return; /* error */
}
```

TipcSrvConnRead

Name	TipcSrvConnRead — read all available data from the connection and queue messages in priority order
Synopsis	<pre>T_BOOL TipcSrvConnRead(srv, timeout) T_IPC_SRV srv; T_REAL8 timeout;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>timeout</i> — maximum number of seconds to wait for data to arrive
Return Values	TRUE if data was successfully read from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnRead fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_VAL_TOO_SMALL — <i>timeout</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnRead
Description	TipcSrvConnRead reads and queues all available messages from the connection to RTserver by calling TipcConnRead. See TipcConnRead for more information on reading data from a connection. TipcSrvConnRead may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnRead sleeps for <i>timeout</i> seconds and returns FALSE. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	TipcSrvConnRead is a low-level function that is normally not used directly by developers; TipcSrvConnRead is normally called only by TipcSrvConnMsgNext and TipcSrvConnMsgSearch.
See Also	TipcConnRead, TipcSrvConnCheck, TipcSrvConnReadCbCreate
Examples	This example reads all data that is immediately available from the connection to RTserver with TipcSrvConnRead: <pre>if (!TipcSrvConnRead(srv, 0.0)) { return; /* error */ }</pre>

TipcSrvConnReadCbCreate

Name	TipcSrvConnReadCbCreate — create a read callback in the connection
Synopsis	<pre>T_CB TipcSrvConnReadCbCreate(<i>srv</i>, <i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_IPC_MT <i>mt</i>; T_IPC_CONN_READ_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>mt</i> — message type to create callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnReadCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnReadCbCreate
Description	TipcSrvConnReadCbCreate creates a read callback in the connection to RTserver by calling TipcConnReadCbCreate. These callbacks are called when a message is read from the connection to RTserver. See TipcConnReadCbCreate for more information on read callbacks. TipcSrvConnReadCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnReadCbCreate creates a read callback in the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcConnReadCbCreate, TipcSrvConnReadCbLookup

Examples

This example creates a read callback in the connection to RTserver for TIME messages which compares the current data time to the data time in the message. If the two times differ by more than 100, a warning is printed, which provides a simple way for an RTclient to detect that it is falling behind:

```
/*===== */
/*..my_srv_read_cb -- compare TIME message against current time */
void T_ENTRY my_srv_read_cb(conn, data, arg)
T_IPC_CONN conn; /* connection to RTserver */
T_IPC_CONN_READ_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_IPC_MSG msg = data->msg;
    T_REAL8 new_time;

    if (!TipcMsgSetCurrent(msg, 0)) {
        return; /* error */
    }

    if (!TipcMsgNextReal8(msg, &new_time)) {
        return; /* error */
    }

    /* compare the new time to the current time */
    if (new_time - TutGetCurrentTime() > 100.0) {
        TutWarning("We appear to be %s seconds behind!\n",
                   TutRealToStr(new_time - TutGetCurrentTime()));
    }
} /* my_srv_read_cb */

/*===== */
/*...code from main program is below */

mt = TipcMtLookupByNum(T_MT_TIME);
if (mt == NULL) {
    return; /* error */
}

if (TipcSrvConnReadCbCreate(srv, mt, my_srv_read_cb, NULL) == NULL)
{
    return; /* error */
}
```

TipcSrvConnReadCbLookup

Name	TipcSrvConnReadCbLookup — look up a read callback in the connection
Synopsis	<pre>T_CB TipcSrvConnReadCbLookup(srv, mt, func, arg) T_IPC_SRV srv; T_IPC_MT mt; T_IPC_CONN_READ_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>mt</i> — message type to look up callback for (null means global callback) <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnReadCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnReadCbLookup
Description	TipcSrvConnReadCbLookup looks up a read callback with the desired message type, function, and argument in the connection to RTserver by calling TipcConnReadCbLookup. These callbacks are called when a message is read from the connection to RTserver. See TipcConnReadCbCreate on connection read callbacks. TipcSrvConnReadCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnReadCbLookup looks up a read callback in the warm connection. See for more information TipcSrvConnOpen on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnReadCbCreate, TipcSrvConnReadCbCreate, TipcSrvConnReadCbLookup; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This code looks up and destroys a global read callback in the connection to RTserver:

```
T_CB cb;

cb = TipcSrvConnReadCbLookup(srv, NULL, my_global_read_cb, NULL);
if (cb == NULL) {
    return; /* error */
}
if (!TutCbDestroy(cb)) {
    return; /* error */
}
```

TipcSrvConnSetAutoFlushSize

Name	TipcSrvConnSetAutoFlushSize — set the auto flush size of the connection
Synopsis	<pre>T_BOOL TipcSrvConnSetAutoFlushSize(srv, auto_flush_size) T_IPC_SRV srv; T_INT4 auto_flush_size;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>auto_flush_size</i> — auto flush size (in bytes)
Return Values	TRUE if the auto flush size was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSetAutoFlushSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_VAL_TOO_SMALL — <i>auto_flush_size</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnSetAutoFlushSize
Description	TipcSrvConnSetAutoFlushSize sets the auto flush size of the connection to RTserver by calling TipcConnSetAutoFlushSize. The auto flush size is used to control how much outgoing data (in bytes) is buffered before being automatically flushed to RTserver. The default for the auto flush size is 8192 bytes. See TipcConnGetAutoFlushSize for more information on auto flush sizes. TipcSrvConnSetAutoFlushSize may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnSetAutoFlushSize sets the auto flush size of the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	Unlike with other connections, the auto flush size T_IPC_NO_AUTO_FLUSH_SIZE cannot be used with the connection to RTserver to disable all automatic flushing of outgoing messages.
See Also	TipcConnGetAutoFlushSize, TipcConnSetAutoFlushSize, TipcSrvConnGetAutoFlushSize

Examples This example creates a connection to RTserver, then sets the auto flush size to 4096 bytes:

```
if (!TipcSrvConnSetAutoFlushSize(srv, 4096)) {  
    return; /* error */  
}
```

TipcSrvConnSetCredentials

Name	TipcSrvConnSetCredentials — set the credentials of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnSetCredentials(srv, auth_policy_id, auth_data, auth_data_len) T_IPC_SRV srv; T_INT4 auth_policy_id; T_PTR auth_data; T_INT4 auth_data_len;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>auth_policy_id</i> — authentication policy identifier <i>auth_data</i> — authentication data specified by the authentication policy identifier <i>auth_data_len</i> — size of the authentication data (in bytes)
Return Values	TRUE if the credentials were successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSetCredentials fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>auth_data</i> or <i>srv</i> was null • T_ERR_VAL_TOO_SMALL — <i>auth_data_len</i> was negative • T_ERR_SRV_ALREADY_CONNECTED — a full connection to RTserver was already established
Description	TipcSrvConnSetCredentials sets the credentials of the connection to RTserver. The credentials are used to authenticate the connection. If authentication fails, an appropriate error code is set when TipcSrvConnOpen is called. Calling TutErrNumGet returns the error code.
Caution	None
See Also	TipcSrvConnOpen

Examples This example sets the RTclient's credentials and attempts to make a connection to the RTserver:

```
#define AUTH_BASIC 1

T_STR user_pass = "jdoe:387203";

if (!TipcSrvConnSetCredentials(srv, AUTH_BASIC, user_pass,
strlen(user_pass)) {
    return; /* error */
}

if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) {
    return; /* error */
}
```

TipcSrvConnSetGmdDirName

Name	TipcSrvConnSetGmdDirName — set the GMD area sub-directory name of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnSetGmdDirName(srv, gmd_dir_name) T_IPC_SRV srv; T_STR gmd_dir_name;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>gmd_dir_name</i> — sub-directory name of the connection to RTserver's GMD area
Return Values	TRUE if the GMD area sub-directory name was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSetGmdDirName fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_IN_USE — the GMD area files are already open • T_ERR_NULL_PTR — <i>srv</i> or <i>gmd_dir_name</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnSetGmdDirName is used to set the GMD area sub-directory file name to a unique value for each connection to RTserver. By default the GMD area sub-directory name is the RTclient's unique subject. However, file conflicts occur when an RTclient process using multiple connections and file-based GMD shares the same unique subject and project on each connection. Multiple connections sharing the same unique subject and project are only allowed when each connection connects to a different RTserver cloud. In that case, TipcSrvConnSetGmdDirName must be called to set the GMD area sub-directory to a unique value for each connection.
Caution	None
See Also	TipcSrvConnGetGmdDirName
Examples	This example sets the GMD area sub-directory of the connection to RTserver: <pre>if (!TipcSrvConnSetGmdDirName(srv, "connection1")) { return; /* error */ }</pre>

TipcSrvConnSetGmdMaxSize

Name	TipcSrvConnSetGmdMaxSize — set the GMD area maximum size of the connection
Synopsis	<pre>T_BOOL TipcSrvConnSetGmdMaxSize(srv, gmd_max_size) T_IPC_SRV srv; T_UINT4 gmd_max_size;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>gmd_max_size</i> — connection to RTserver GMD area maximum size (in bytes)
Return Values	TRUE if the GMD area maximum size was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSetGmdMaxSize fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created • any error number from TipcConnSetGmdMaxSize
Description	TipcSrvConnSetGmdMaxSize sets the GMD area maximum size in bytes of the connection to RTserver by calling TipcConnSetGmdMaxSize. A GMD area holds guaranteed message delivery information for incoming and outgoing messages. There are two types of guaranteed message delivery: file-based and memory-only. See TipcConnGmdFileCreate for more information on GMD areas. For file-based GMD this limit is the maximum file size, and for memory-only GMD this limit is the maximum amount of memory allowed. The default GMD area maximum size is 0, which means that no GMD area maximum size limit checking is performed. The connection GMD area maximum size and the connection delivery timeout can be used to constrain the amount of system resources used for GMD by space or by time. If the connection GMD area maximum size is exceeded, no further messages can be sent with guaranteed delivery (TipcSrvConnMsgSend returns FALSE and sets the global SmartSockets error number to T_ERR_VAL_TOO_LARGE) until some unacknowledged previously sent messages are acknowledged.

TipcSrvConnSetGmdMaxSize may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnSetGmdMaxSize sets the GMD area maximum size of the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution None

See Also TipcConnSetGmdMaxSize, TipcSrvConnGmdFileCreate, TipcSrvConnGetGmdMaxSize

Examples This example sets the GMD area maximum size of the connection to RTserver to one million bytes:

```
if (!TipcSrvConnSetGmdMaxSize(srv, 1000000)) {
    return; /* error */
}
```

TipcSrvConnSetProject

Name	TipcSrvConnSetProject — set the project of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnSetProject(<i>srv</i>, <i>project</i>) T_IPC_SRV <i>srv</i>; T_STR <i>project</i>;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>project</i> — project to set
Return Values	TRUE if the project was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSetProject fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>project</i> was null • T_ERR_SRV_ALREADY_CONNECTED — a full connection to RTserver was already established
Description	TipcSrvConnSetProject sets the project of the connection to RTserver.
Caution	None
See Also	TipcSrvConnCreate, TipcSrvConnGetProject
Examples	This example sets the RTclient's project: <pre>if (!TipcSrvConnSetProject(<i>srv</i>, "development")) { return; /* error */ }</pre>

TipcSrvConnSetServerNames

Name	TipcSrvConnSetServerNames — set the server names of a connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnSetServerNames(<i>srv</i>, <i>server_names</i>) T_IPC_SRV <i>srv</i>; T_STR <i>server_names</i>;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>server_names</i> — comma delimited string of server names
Return Values	TRUE if the server names were successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSetServerNames fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>server_names</i> was null • T_ERR_SRV_ALREADY_CONNECTED — a full connection to RTserver was already established
Description	TipcSrvConnSetServerNames sets the server names of the connection to RTserver.
Caution	None
See Also	TipcSrvConnCreate, TipcSrvConnGetServerNamesList
Examples	This example sets the RTclient’s server names: <pre>if (!TipcSrvConnSetServerNames(<i>srv</i>, "tcp:moe, tcp:larry, tcp:curly")) { return; /* error */ }</pre>

TipcSrvConnSetServerNamesList

Name	TipcSrvConnSetServerNamesList — set the server names of the connection to RTserver in a linked list of strings
Synopsis	<pre>T_BOOL TipcSrvConnSetServerNamesList(srv, server_names_list) T_IPC_SRV srv; T_STR_LIST server_names_list;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>server_names_list</i> — string list of server names
Return Values	TRUE if the server names were successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSetServerNamesList fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>server_names_list</i> was null • T_ERR_SRV_ALREADY_CONNECTED — a full connection to RTserver was already established
Description	TipcSrvConnSetServerNamesList sets the server names of the connection to RTserver in a linked list of strings. The string list values are copied to internal storage, so the caller does not need to maintain storage for them.
Caution	None
See Also	TipcSrvConnCreate, TipcSrvConnGetServerNamesList
Examples	This example sets the RTclient's server names: <pre>T_OPTION option; T_STR_LIST server_names; if (NULL == (option = TutOptionLookup("server_names"))) { return; /* error */ } if (!TutOptionGetStrList(option, &server_names)) { return; /* error */ } if (!TipcSrvConnSetServerNamesList(srv, server_names)) { return; /* error */ }</pre>

TipcSrvConnSetTimeout

Name	TipcSrvConnSetTimeout — set a timeout property of the connection
Synopsis	<pre>T_BOOL TipcSrvConnSetTimeout(<i>srv</i>, <i>timeout</i>, <i>value</i>) T_IPC_SRV <i>srv</i>; T_IPC_TIMEOUT <i>timeout</i>; T_REAL8 <i>value</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>timeout</i> — which timeout property to set <i>value</i> — <i>timeout</i> value (in seconds)
Return Values	TRUE if the timeout was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSetTimeout fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_VAL_INVALID — <i>timeout</i> was not a valid T_IPC_TIMEOUT • T_ERR_VAL_TOO_SMALL — <i>value</i> was negative • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnSetTimeout sets a timeout property of the connection to RTserver by setting one of the options: Server_Read_Timeout, Server_Write_Timeout, Server_Keep_Alive_Timeout, or Server_Delivery_Timeout. The timeout properties control how often the connection to RTserver is checked for possible network failures. See TipcSrvConnGetTimeout for more information on connection timeouts. TipcSrvConnSetTimeout may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnSetTimeout sets a timeout property of the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	None
See Also	TipcConnSetTimeout, TipcSrvConnGetTimeout

Examples This example sets the read timeout of the connection to RTserver with TipcSrvConnSetTimeout to 10 seconds:

```
if (!TipcSrvConnSetTimeout(srv, T_IPC_TIMEOUT_READ, 10.0)) {  
    return; /* error */  
}
```

An alternate way to set the read timeout of the connection to RTserver is to set the option `Server_Read_Timeout`:

```
TutCommandParseStr("setopt server_read_timeout 10.0");
```

TipcSrvConnSetUsernamePassword

Name	TipcSrvConnSetUsernamePassword — set the basic credentials of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnSetUsernamePassword(srv, username, password) T_IPC_SRV srv; T_STR username; T_STR password;</pre>
Arguments	<i>srv</i> — the connection to RTserver <i>username</i> — username to use for basic credentials <i>password</i> — password to use for basic credentials
Return Values	TRUE if the basic credentials were successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSetUsernamePassword fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i>, <i>username</i> or <i>password</i> was null • T_ERR_SRV_ALREADY_CONNECTED — a connection to RTserver has already been established
Description	TipcSrvConnSetUsernamePassword sets the basic credentials of the connection to RTserver. When RTserver has enabled Basic Security, the basic credentials are used to authenticate the connection. If authentication fails, an error code is set when TipcSrvCreate is called. For more information on Basic Security, see the <i>TIBCO SmartSockets User’s Guide</i>.
See Also	TipcSrvConnOpen
Examples	This example sets the connection’s basic credentials and attempts to make a connection to the RTserver: <pre>if (!TipcSrvConnSetUsernamePassword(srv, argv[1], argv[2])) { return; /* error */ } if (!TipcSrvConnOpen(srv, T_IPC_SRV_CONN_FULL)) { return; /* error */ }</pre>

TipcSrvConnStdSubjectSetSubscribe

Name	TipcSrvConnStdSubjectSetSubscribe — start or stop subscribing to all standard subjects
Synopsis	<pre>T_BOOL TipcSrvConnStdSubjectSetSubscribe(srv, subscribe_status) T_IPC_SRV srv; T_BOOL subscribe_status;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>subscribe_status</i> — whether to start or stop subscribing to all standard subjects
Return Values	TRUE if the subscribe statuses of the standard subjects were successfully set, FALSE otherwise.
Diagnostics	If TipcSrvConnStdSubjectSetSubscribe fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_VAL_INVALID — <i>subscribe_status</i> was not TRUE or FALSE • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnStdSubjectSetSubscribe sets the subscribe statuses of the standard subjects by calling TipcSrvConnSubjectSetSubscribe for each standard subject. TipcSrvConnStdSubjectSetSubscribe is often used to subscribe to subjects listed in a command file. There are many standard subjects: • subjects listed in the option Subjects • <i>_node</i> — the node-specific subject, such as <i>workstation1</i> • <i>_all</i> — common subject for all RTclients • standard process type subject, such as <i>_mon</i>. User-defined RTclients do not have a standard process type subject. RTmon uses <i>_mon</i>. As each standard subject subscribe status is changed, TipcSrvConnStdSubjectSetSubscribe prints out an informational message with TutOut. TipcSrvConnStdSubjectSetSubscribe also flushes the retrieval messages with TipcSrvConnFlush when it is done.

TpcSrvConnStdSubjectSetSubscribe may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TpcSrvConnStdSubjectSetSubscribe changes the subscribe statuses of the subjects as if the RTclient were still connected to RTserver. See TpcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution None

See Also TpcSrvConnSubjectSetSubscribe

Examples This example sets the option Subjects and uses TpcSrvConnStdSubjectSetSubscribe to start subscribing to the standard subjects:

```
TutCommandParseStr("setopt subjects subject1, subject2");
if (!TpcSrvConnStdSubjectSetSubscribe(srv, TRUE)) {
    return; /* error */
}
```

The output from the above fragment would be similar to:

```
Start subscribing to subject </subject1>.
Start subscribing to subject </subject2>.
Start subscribing to subject </_workstation1>.
Start subscribing to subject </_all>.
```

TipcSrvConnSubjectCbCreate

Name	TipcSrvConnSubjectCbCreate — create a subject callback
Synopsis	<pre>T_CB TipcSrvConnSubjectCbCreate(<i>srv</i>, <i>subject</i>, <i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_STR <i>subject</i>; T_IPC_MT <i>mt</i>; T_IPC_SRV_SUBJECT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>subject</i> — subject to create callback for (null means all subjects) <i>mt</i> — message type to create callback for (null means all message types) <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnSubjectCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnSubjectCbCreate creates a subject callback. These callbacks are called by TipcSrvConnMsgProcess to process a message when it is received by the <i>subject</i>. If <i>subject</i> is null, then <i>func</i> is called for all messages of type <i>mt</i> received by this RTclient. If <i>mt</i> is null, then <i>func</i> is called for every type of message received by <i>subject</i>. If both <i>subject</i> and <i>mt</i> are null, <i>func</i> is called for every message received by this RTclient. Subject wildcards (* or ...) are supported.

Caution Do not use both subject callbacks and connection process callbacks at the same time. Each callback type represents a different way of partitioning data and they are not orthogonal. Connection process callbacks partition the data according to message type; subject callbacks partition according to subjects. If these types of callbacks are used together, connection process callbacks are always of higher priority. Connection process callbacks are called in priority order until the last connection process callback is called. Then the subject callbacks are called in priority order.

The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.

See Also TipcSrvConnSubjectCbLookup, TipcSrvConnSubjectDefaultCbCreate, TipcSrvConnSubjectDefaultCbLookup; see the *TIBCO SmartSockets Utilities* for information on TutCbSetPriority.

Examples This example creates a subject callback for T_MT_INFO messages that are received by the /stocks subject. A second subject callback is created for all message types received by the /emergency subject. The emergency callback has a higher priority so that it will be called first:

```
/* ===== */
/*..info_subject_cb -- process INFO messages recvd by /stocks */
void T_ENTRY info_subject_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_SRV_SUBJECT_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_STR info_str;
    T_STR subject;

    if (!TipcMsgSetCurrent(data->msg, 0)) {
        return; /* error */
    }

    if (!TipcMsgNextStr(data->msg, &info_str)) {
        return; /* error */
    }

    if (!TipcMsgGetDest(data->msg, &subject)) {
        return; /* error */
    }

    TutOut("Information message received by %s is %s\n", subject,
           info_str);
} /* info_subject_cb */
```



```

/* ===== */
/*..emergency_cb -- process messages recvd by emergency subject */
void T_ENTRY emergency_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_SRV_SUBJECT_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_STR mt_name;
    T_IPC_MT mt;

    if (!TipcMsgGetType(data->msg, &mt)) {
        return; /* error */
    }

    if (!TipcMtGetName(mt, &mt_name)) {
        return; /* error */
    }

    TutOut("WARNING: Emergency message received: %s\n", mt_name);
} /* emergency_cb */

/* ===== */
/*...code from calling function is below */

mt = TipcMtLookupByNum(T_MT_INFO);
if (mt == NULL) {
    return; /* error */
}
cb = TipcSrvConnSubjectCbCreate(srv, "/stocks", mt,
                                info_subject_cb,
                                NULL);
if (cb == NULL) {
    return; /* error */
}
cb = TipcSrvConnSubjectCbCreate(srv, "/emergency", NULL,
                                emergency_cb,
                                NULL);
if (cb == NULL) {
    return; /* error */
}
if (!TutCbSetPriority(cb, 100)) {
    return; /* error */
}
}

```

TipcSrvConnSubjectCbDestroyAll

Name	TipcSrvConnSubjectCbDestroyAll — destroy all subject callbacks
Synopsis	<pre>T_BOOL TipcSrvConnSubjectCbDestroyAll(srv) T_IPC_SRV srv;</pre>
Arguments	<i>srv</i> — connection handle to RTserver
Return Values	TRUE if successful, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectCbDestroyAll fails, it returns NULL and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>srv</i> was null • T_ERR_DOESNT_EXIST — no subject callbacks exist to destroy
Description	TipcSrvConnSubjectCbDestroyAll destroys all registered subject callbacks including default subject callbacks. This is a convenience function that allows for all of the subject callbacks to be destroyed easily.
Caution	None
See Also	TipcSrvConnSubjectCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example destroys all subject callbacks: <pre>if (!TipcSrvConnSubjectCbDestroyAll(srv)) { return; /* error */ }</pre>

TipcSrvConnSubjectCbLookup

Name	TipcSrvConnSubjectCbLookup — look up a subject callback
Synopsis	<pre>T_CB TipcSrvConnSubjectCbLookup(srv, subject, mt, func, arg) T_IPC_SRV srv; T_STR subject; T_IPC_MT mt; T_IPC_SRV_SUBJECT_CB_FUNC func; T_CB_ARG arg;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>subject</i> — subject to look up callback for <i>mt</i> — message type to look up callback for <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnSubjectCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnSubjectCbLookup looks up a subject callback with the desired subject, message type, function, and argument. When the subject contains wildcards, the wildcarded subject must exactly match the subject that was used when creating the subject callback. If successful, TipcSrvConnSubjectCbLookup returns one and only one subject callback.
Caution	None
See Also	TipcSrvConnSubjectCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.

Examples This example looks up a global subject callback for the /emergency subject:

```
T_CB cb;

cb = TIPC_SrvConnSubjectCbLookup(srv, "/emergency", NULL,
emergency_cb,
                                NULL);

if (cb == NULL) {
    return; /* error */
}
```

TipcSrvConnSubjectDefaultCbCreate

Name	TipcSrvConnSubjectDefaultCbCreate — create a default subject callback
Synopsis	<pre>T_CB TipcSrvConnSubjectDefaultCbCreate(<i>srv</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_IPC_SRV_SUBJECT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>func</i> — callback function <i>arg</i> — user-defined argument to pass to <i>func</i>
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnSubjectDefaultCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnSubjectDefaultCbCreate creates a default subject callback. Callbacks are called by TipcSrvConnMsgProcess to perform the processing of a message. The callback function, <i>func</i> , is only called when no other subject callback matches the subject of the message being processed.
Caution	The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.
See Also	TipcSrvConnSubjectCbCreate, TipcSrvConnSubjectCbLookup, TipcSrvConnSubjectDefaultCbLookup

Examples This example creates a default subject callback:

```

/*===== */
/*..default_subject_cb -- default subject callback */
void T_ENTRY default_subject_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_SRV_SUBJECT_CB_DATA data;
T_CB_ARG arg; /* not used */
{
    T_STR mt_name;
    T_IPC_MT mt;
    T_STR subject;

    if (!TipcMsgGetType(data->msg, &mt)) {
        return; /* error */
    }

    if (!TipcMtGetName(mt, &mt_name)) {
        return; /* error */
    }

    if (!TipcMsgGetDest(data->msg, &subject));
        return; /* error */
    }

    TutOut("DEFAULT: Message type %s recvd by %s\n",
           mt_name, subject);
} /* default_subject_cb */

/*===== */
/*...code from calling function is below */

cb = TipcSrvConnSubjectDefaultCbCreate(srv, default_subject_cb,
                                       NULL);
if (cb == NULL) {
    return; /* error */
}

```

TipcSrvConnSubjectDefaultCbLookup

Name	TipcSrvConnSubjectDefaultCbLookup — look up a default subject callback
Synopsis	<pre>T_CB TipcSrvConnSubjectDefaultCbLookup(<i>srv</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_IPC_SRV_SUBJECT_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>func</i> — callback function <i>arg</i> — user-defined argument
Return Values	New callback if successful, NULL otherwise.
Diagnostics	If TipcSrvConnSubjectDefaultCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnSubjectDefaultCbLookup looks up a default subject callback with the desired function and argument. See TipcSrvConnSubjectDefaultCbCreate for more information on default subject callbacks.
Caution	None
See Also	TipcSrvConnSubjectDefaultCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.
Examples	This example looks up a default subject callback: <pre>T_CB cb; cb = TipcSrvConnSubjectDefaultCbLookup(<i>srv</i>, default_subject_cb, NULL); if (cb == NULL) { return; /* error */ }</pre>

TipcSrvConnSubjectGetDefaultPrefix

Name	TipcSrvConnSubjectGetDefaultPrefix — get the default subject prefix of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnSubjectGetDefaultPrefix(srv, default_prefix_return) T_IPC_SRV srv; T_STR *default_prefix_return;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>default_prefix_return</i> — returned default subject prefix
Return Values	TRUE if the default subject prefix was successfully retrieved for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectGetDefaultPrefix fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>srv</i> or <i>default_prefix_return</i> was null
Description	TipcSrvConnSubjectGetDefaultPrefix gets the default subject prefix of the connection to RTserver.
Caution	The storage pointed to by <i>default_prefix_return</i> should not be modified or freed. The storage is thread safe, but may be overwritten by subsequent calls to SmartSockets APIs within the same thread. If you wish to preserve the value, you should make a copy of the value.
See Also	TipcSrvConnSubjectSetDefaultPrefix
Examples	This example gets the RTclient’s default subject prefix: <pre>T_STR default_prefix; if (!TipcSrvConnSubjectGetDefaultPrefix(srv, &default_prefix)) { return; /* error */ }</pre>

TipcSrvConnSubjectGetSubscribe

Name	TipcSrvConnSubjectGetSubscribe — determine if an RTclient is subscribing to a subject
Synopsis	<pre>T_BOOL TipcSrvConnSubjectGetSubscribe(<i>srv</i>, <i>subject</i>, <i>subscribe_status_return</i>) T_IPC_SRV <i>srv</i>; T_STR <i>subject</i>; T_BOOL *<i>subscribe_status_return</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>subject</i> — name of the subject to get subscribe status for <i>subscribe_status_return</i> — storage for subscribe status
Return Values	TRUE if the subscribe status of the subject was successfully retrieved, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectGetSubscribe fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i>, <i>subject</i>, or <i>subscribe_status_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnSubjectGetSubscribe determines whether an RTclient is subscribing to a subject. The subscribe status controls whether the RTclient receives messages published (sent) to that subject. If the subscribe status of <i>subject</i> is TRUE, the RTclient receives messages with <i>subject</i> as their destination that are sent to RTserver. To start or stop subscribing to a subject, use TipcSrvConnSubjectSetSubscribe. To find out if an RTclient is subscribing to a subject, including load balancing information, use TipcSrvConnSubjectGetSubscribeLb. TipcSrvConnSubjectGetSubscribe may attempt to automatically create a connection to RTserver. TipcSrvConnSubjectGetSubscribe gets the subscribe status of the subject for both full and warm RTclient connections. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
Caution	If TipcSrvConnSubjectGetSubscribe returns FALSE, it does not store a value in <i>subscribe_status_return</i> .
See Also	TipcMonClientSubscribeSetWatch, TipcMonSubjectSubscribeSetWatch, TipcSrvConnSubjectSetSubscribe, TipcSrvConnSubjectGetSubscribeLb

Examples This example gets the subscribe status of the subject /system/thermal:

```
T_BOOL subscribe_status;
if (!TipcSrvConnSubjectGetSubscribe(srv, "/system/thermal",
                                     &subscribe_status)) {
    return; /* error */
}
TutOut("This process is %s the /system/thermal subject.\n",
       subscribe_status ? "subscribing to" : "not subscribing to");
```

TipcSrvConnSubjectGetSubscribeLb

Name	TipcSrvConnSubjectGetSubscribeLb — determine if an RTclient is subscribing to a subject, including load balancing information
Synopsis	<pre>T_BOOL TipcSrvConnSubjectGetSubscribeLb(srv, subject, subscribe_status_return, lb_status_return) T_IPC_SRV srv; T_STR subject; T_BOOL *subscribe_status_return; T_BOOL *lb_status_return;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>subject</i> — name of the subject to get subscribe status for <i>subscribe_status_return</i> — storage for subscribe status <i>lb_status_return</i> — storage for load balancing status
Return Values	TRUE if the subscribe statuses of the subject were successfully retrieved, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectGetSubscribeLb fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i>, <i>subject</i>, <i>subscribe_status_return</i>, or <i>lb_status_return</i> was null • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnSubjectGetSubscribeLb and TipcSrvConnSubjectSetSubscribeLb are variants of TipcSrvConnSubjectGetSubscribe and TipcSrvConnSubjectSetSubscribe that include load balancing control. TipcSrvConnSubjectGetSubscribeLb determines whether or not an RTclient is subscribing to a subject, including load balancing information. The subscribe status controls whether the RTclient receives messages published (sent) to that subject. If the subscribe status of subject is TRUE, then the RTclient receives messages with <i>subject</i> as their destination that are sent to RTserver. To start or stop subscribing to a subject, with or without load balancing, use TipcSrvConnSubjectSetSubscribeLb. If the load balancing status of subject is TRUE, then the RTclient is part of the load balancing calculation. Otherwise, the RTclient overrides load balancing and always receives the messages regardless of whether or not the message was published with load balancing.

TipcSrvConnSubjectGetSubscribeLb may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnSubjectGetSubscribeLb gets the subscribe status of the subject as if the RTclient were still connected to RTserver. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

Caution If TipcSrvConnSubjectGetSubscribeLb returns FALSE, it does not store a value in *subscribe_status_return* or *lb_status_return*.

See Also TipcMonClientSubscribeSetWatch, TipcMonSubjectSubscribeSetWatch, TipcSrvConnSubjectGetSubscribe, TipcSrvConnSubjectSetSubscribeLb

Examples This example uses TipcSrvConnSubjectGetSubscribeLb to get the subscribe status of the subject /system/thermal:

```
T_BOOL subscribe_status;
T_BOOL lb_status;

if (!TipcSrvConnSubjectGetSubscribeLb(srv, "/system/thermal",
                                     &subscribe_status, &lb_status))
{
    return; /* error */
}

TutOut("This process is %s the /system/thermal subject.\n",
       subscribe_status ? "subscribing to" : "not subscribing to");

if (subscribe_status) {
    TutOut("This process is %s load balancing.\n",
          lb_status ? "part of" : "not part of");
}
```

TipcSrvConnSubjectGetUnique

Name	TipcSrvConnSubjectGetUnique — get the unique subject of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnSubjectGetUnique(srv, unique_subject_return) T_IPC_SRV srv; T_STR *unique_subject_return;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>unique_subject_return</i> — returned unique subject
Return Values	TRUE if the unique subject was successfully retrieved from the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectGetUnique fails, it returns FALSE and sets the global SmartSockets error number to: • T_ERR_NULL_PTR — <i>srv</i> or <i>unique_subject_return</i> was null
Description	TipcSrvConnSubjectGetUnique gets the unique subject of the connection to RTserver.
Caution	None
See Also	TipcSrvConnSubjectSetUnique
Examples	This example gets the RTclient's unique subject: <pre>T_STR unique_subject; if (!TipcSrvConnSubjectGetUnique(srv, &unique_subject)) { return; /* error */ }</pre>

TipcSrvConnSubjectGmdInit

Name	TipcSrvConnSubjectGmdInit — initialize GMD accounting in RTserver for a subject to which messages will be published
Synopsis	<pre>T_BOOL TipcSrvConnSubjectGmdInit(srv, subject) T_IPC_SRV srv; T_STR subject;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>subject</i> — name of subject to which messages will be published with GMD
Return Values	TRUE if the request for GMD initialization was successfully initiated, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectGmdInit fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>subject</i> was null • any error number from TipcSrvConnMsgWrite
Description	TipcSrvConnSubjectGmdInit initializes GMD accounting in RTserver for a subject to which messages will be published by sending a GMD_INIT_CALL message to RTserver. RTserver responds by sending back a GMD_INIT_RESULT message when GMD initialization is complete. The response should come back very quickly. The GMD_INIT_RESULT message contains one field: a STR field containing the subject name for which initialization has completed. The polling RTclient can use functions such as TipcSrvConnMsgSearchType to get the response. Both guaranteed message delivery and load balancing currently use the same accounting mechanism. Use TipcSrvConnSubjectLbInit, however, to initialize load balancing accounting for a subject. When the RTserver connected to the publishing RTclient routes a message for GMD, it keeps track of which RTclients should receive the message and the status of GMD to each of these RTclients. If this RTserver is not already tracking GMD for <i>subject</i> when the first published message arrives from RTclient, the RTserver must start a monitoring-like watch mechanism between the RTservers of all subscribing RTclients. This increases the scalability of GMD because only the relevant RTservers dynamically exchange GMD information. The GMD accounting continues until no messages have been published with GMD or load balancing for an interval specified by the Gmd_Publish_Timeout option. The default for Gmd_Publish_Timeout is 300 seconds.

This GMD accounting starts very quickly, but it is not instantaneous. `TipcSrvConnSubjectGmdInit` is intended to be used by advanced publish-subscribe applications that wish to ensure that RTserver is primed and immediately ready for GMD publish-subscribe routing.

`TipcSrvConnSubjectGmdInit` is usually only used at initialization, but it can be called at any time by advanced applications that wish to initialize GMD publishing accounting. The use of `TipcSrvConnSubjectGmdInit` is strictly optional for GMD publishing; if it is not called, the GMD setup occurs when the first GMD message is published to *subject*.

If RTserver is already tracking GMD for *subject*, RTserver immediately returns a `GMD_INIT_RESULT` message.

Caution `TipcSrvConnSubjectGmdInit` sends a `GMD_INIT_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

See Also `TipcSrvConnMsgSend`, `TipcSrvConnMsgWrite`, `TipcSrvConnSubjectLbInit`

Examples This example uses `TipcSrvConnSubjectGmdInit` to initialize GMD accounting in RTserver for the `/system/thermal` subject and then waits for the result message:

```
T_IPC_MT mt;
T_IPC_MSG result_msg;

/* initialize GMD accounting in RTserver */
if (!TipcSrvConnSubjectGmdInit(srv, "/system/thermal")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_GMD_INIT_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait a short time for the GMD init result */
result_msg = TipcSrvConnMsgSearchType(srv, 10.0, mt);
if (result_msg == NULL) {
    return; /* error */
}

/* must set current field first */
if (!TipcMsgSetCurrent(result_msg, 0)) {
    return; /* error */
}

/* at this point, we could read the first field of the result message to verify the subject name */
```

TipcSrvConnSubjectLbInit

Name	TipcSrvConnSubjectLbInit — initialize load balancing accounting in RTserver for a subject to which messages will be published
Synopsis	<pre>T_BOOL TipcSrvConnSubjectLbInit(srv, subject) T_IPC_SRV srv; T_STR subject;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>subject</i> — name of subject to which messages will be published with load balancing
Return Values	TRUE if the request for load balancing initialization was successfully initiated, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectLbInit fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>subject</i> was null • any error number from TipcSrvConnMsgWrite
Description	TipcSrvConnSubjectLbInit initializes load balancing accounting in RTserver for a subject to which messages will be published by sending a GMD_INIT_CALL message to RTserver. RTserver responds by sending back a GMD_INIT_RESULT message when load balancing initialization is complete. The response should come back very quickly. The GMD_INIT_RESULT message contains one field: a STR string field containing the subject name for which initialization has completed. The polling RTclient can use functions such as TipcSrvConnMsgSearchType to get the response. Both guaranteed message delivery and load balancing currently use the same accounting mechanism. Use TipcSrvConnSubjectLbInit, however, to initialize load balancing accounting for a subject. When the RTserver connected to the publishing RTclient routes a message for load balancing, it keeps track of which RTclients should receive the message. If this RTserver is not already tracking load balancing for <i>subject</i> when the first published message arrives from RTclient, the RTserver must start a monitoring-like watch mechanism between the RTservers of all subscribing RTclients. This increases the scalability of load balancing because only the relevant RTservers dynamically exchange load balancing information. The load balancing accounting continues until no messages have been published with GMD or load balancing for an interval specified by the Gmd_Publish_Timeout option. The default for Gmd_Publish_Timeout is 300 seconds.

This load balancing accounting starts very quickly, but it is not instantaneous. `TipcSrvConnSubjectLbInit` is intended to be used by advanced publish-subscribe applications that wish to ensure that RTserver is primed and immediately ready for load balancing publish-subscribe routing.

`TipcSrvConnSubjectLbInit` is usually only used at initialization, but it can be called at any time by advanced applications which wish to initialize load balancing publishing accounting. The use of `TipcSrvConnSubjectLbInit` is strictly optional for load balancing publishing; if it is not called, the load balancing setup occurs when the first GMD or load balancing message is published to *subject*.

If RTserver is already tracking load balancing for *subject*, RTserver immediately returns a `GMD_INIT_RESULT` message.

Caution `TipcSrvConnSubjectLbInit` sends a `GMD_INIT_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

See Also `TipcSrvConnSubjectLbInit`, `TipcSrvConnMsgSend`, `TipcSrvConnMsgWrite`

Examples This example uses `TipcSrvConnSubjectLbInit` to initialize load balancing accounting in RTserver for the `/system/thermal` subject and then waits for the result message:

```
T_IPC_MT mt;
T_IPC_MSG result_msg;

/* initialize load balancing accounting in RTserver */
if (!TipcSrvConnSubjectLbInit(srv, "/system/thermal")) {
    return; /* error */
}

mt = TipcMtLookupByNum(T_MT_GMD_INIT_RESULT);
if (mt == NULL) {
    return; /* error */
}

/* wait a short time for the GMD init result */
result_msg = TipcSrvConnMsgSearchType(srv, 10.0, mt);
if (result_msg == NULL) {
    return; /* error */
}

/* must set current field first */
if (!TipcMsgSetCurrent(result_msg, 0)) {
    return; /* error */
}

/* at this point, we could read the first field of the result message to verify the subject name */
```

TipcSrvConnSubjectSetDefaultPrefix

Name	TipcSrvConnSubjectSetDefaultPrefix — set the default subject prefix of the connection to RTserver
Synopsis	<pre>T_BOOL TipcSrvConnSubjectSetDefaultPrefix(srv, default_prefix) T_IPC_SRV srv; T_STR default_prefix;</pre>
Arguments	<i>srv</i> — connection to RTserver <i>default_prefix</i> — default subject prefix to set
Return Values	TRUE if the default subject prefix was successfully set for the connection to RTserver, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectSetDefaultPrefix fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>default_prefix</i> was null • T_ERR_VAL_INVALID — invalid value for <i>default_prefix</i> • T_ERR_SRV_ALREADY_CONNECTED — a full or warm connection to RTserver was already established
Description	TipcSrvConnSubjectSetDefaultPrefix sets the default subject prefix of the connection to RTserver.
Caution	None
See Also	TipcSrvConnCreate, TipcSrvConnSubjectGetDefaultPrefix
Examples	This example sets the RTclient’s default subject prefix: <pre>if (!TipcSrvConnSubjectSetDefaultPrefix(srv, "/client")) { return; /* error */ }</pre>

TipcSrvConnSubjectSetSubscribe

Name	TipcSrvConnSubjectSetSubscribe — start or stop subscribing to a subject
Synopsis	<pre>T_BOOL TipcSrvConnSubjectSetSubscribe(<i>srv</i>, <i>subject</i>, <i>subscribe_status</i>) T_IPC_SRV <i>srv</i>; T_STR <i>subject</i>; T_BOOL <i>subscribe_status</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>subject</i> — name of the subject to set subscribe status for <i>subscribe_status</i> — subscribe status
Return Values	TRUE if the subscribe status of the subject was successfully set, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectSetSubscribe fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>subject</i> was null • T_ERR_VAL_INVALID — one of: — <i>subscribe_status</i> was not TRUE or FALSE, or — <i>subject</i> was the unique subject and <i>subscribe_status</i> was FALSE (RTclient cannot stop subscribing to its unique subject), or — <i>subject</i> was not a valid subject • T_ERR_VAL_SAME — <i>subscribe_status</i> is the same as the current subscribe status • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created
Description	TipcSrvConnSubjectSetSubscribe sets whether or not this RTclient is subscribing to a subject. The subscribe status controls whether or not the RTclient receives messages published (sent) to that subject. If the subscribe status of subject is TRUE, then the RTclient receives messages with <i>subject</i> as their destination that are sent to RTserver. To determine if an RTclient is subscribing to a subject, use TipcSrvConnSubjectGetSubscribe.

TpcSrvConnSubjectSetSubscribe may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TpcSrvConnSubjectSetSubscribe sets the subscribe status of the subject as if the RTclient were still connected to RTserver. See TpcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.

There are a number of standard subjects available to all RTclients. See TpcSrvConnStdSubjectSetSubscribe for more information on these standard subjects.

To determine if an RTclient is subscribing to a subject, with or without load balancing information, use TpcSrvConnSubjectSetSubscribeLb.

Caution TpcSrvConnSubjectSetSubscribe sends a SUBJECT_SET_SUBSCRIBE message to RTserver, but does not explicitly flush the message. See TpcSrvConnGetAutoFlushSize for more information on message buffering.

See Also TpcMonClientSubscribeSetWatch, TpcMonSubjectSubscribeSetWatch, TpcSrvConnSubjectGetSubscribe, TpcSrvConnStdSubjectSetSubscribe, TpcSrvConnSubjectSetSubscribeLb

Examples This example uses TpcSrvConnSubjectSetSubscribe to start subscribing to the subject /system/thermal:

```
if (!TpcSrvConnSubjectSetSubscribe(srv, "/system/thermal", TRUE))
{
    return; /* error */
}
```

TipcSrvConnSubjectSetSubscribeEx

Name	TipcSrvConnSubjectSetSubscribeEx — start or stop subscribing to a subject using extended options
Synopsis	<pre>T_BOOL TipcSrvConnSubjectSetSubscribeEx(srv, subj_name, status, type, flags, type_specific_struct) T_IPC_SRV srv; T_STR subj_name; T_BOOL status; T_INT4 type; T_INT4 flags; T_PTR type_specific_struct;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>subj_name</i> — name of the subject to set subscribe status for <i>status</i> — subscribe status (TRUE to subscribe or FALSE to unsubscribe) <i>type</i> — type of subscribe to use (T_SUBJ_NONE for a normal subscribe or T_SUBJ_CACHE to enable the SmartSockets Cache interface) <i>flags</i> — describe the attribute of <i>type</i> <i>type_specific_struct</i> — used to pass any additional information needed by a <i>type</i>
Return Values	TRUE if the subscribe status of the subject was successfully set, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectSetSubscribeEx fails, it returns FALSE and sets the global SmartSockets error number to one of: - T_ERR_DOESNT_EXIST — <i>type</i> was T_SUBJ_CACHE and caching options were not found, usually because caching was not initialized. See the <i>TIBCO SmartSockets Cache User's Guide</i> for more information. - any error number from TipcSrvConnSubjectSetSubscribe.
Description	TipcSrvConnSubjectSetSubscribeEx is used to subscribe to a subject using extended options. Currently the only option available is caching. If <i>type</i> is T_SUBJ_NONE, <i>flags</i> must be set to zero (0). If <i>type</i> is T_SUBJ_CACHE, <i>flags</i> must be one of: - T_SUBJ_FLAG_SC_INITIAL_LOAD — for force initial load - T_SUBJ_FLAG_SC_LATEST_MSG — to retrieve the last message from the cache or a new message if one arrives before the cache retrieval has completed

If type is either T_SUBJ_NONE or T_SUBJ_CACHE, *type_specific_struct* must be set to T_NULL.

Caution None

See Also None

Examples This example retrieves a message from SmartSockets Cache

```
/* Retrieve message from Cache */
TutOut("Retrieving messages from Cache ...\n");

TipcSrvConnSubjectSetSubscribeEx(
    srv,                               /* requested server object */
    "/SCache/*",                       /* requested subject */
    T_TRUE,                            /* true to subscribe */
    T_SUBJ_CACHE,                     /* type of subscribe */
    T_SUBJ_FLAG_SC_INITIAL_LOAD,      /* attribute type to last value */
    T_NULL);                          /* as using T_SUBJ_CACHE */
```

TipcSrvConnSubjectSetSubscribeLb

Name	TipcSrvConnSubjectSetSubscribeLb — start or stop subscribing to a subject, with or without load balancing
Synopsis	<pre>T_BOOL TipcSrvConnSubjectSetSubscribeLb(<i>srv</i>, <i>subject</i>, <i>subscribe_status</i>, <i>lb_status</i>) T_IPC_SRV <i>srv</i>; T_STR <i>subject</i>; T_BOOL <i>subscribe_status</i>; T_BOOL <i>lb_status</i>;</pre>
Arguments	<i>srv</i> — connection handle to RTserver <i>subject</i> — name of the subject to set subscribe status for <i>subscribe_status</i> — subscribe status <i>lb_status</i> — load balancing status
Return Values	TRUE if the subscribe status of the subject was successfully set, FALSE otherwise.
Diagnostics	If TipcSrvConnSubjectSetSubscribeLb fails, it returns FALSE and sets the global SmartSockets error number to one of: • T_ERR_NULL_PTR — <i>srv</i> or <i>subject</i> was null • T_ERR_VAL_INVALID — one of: — <i>subscribe_status</i> or <i>lb_status</i> were not TRUE or FALSE, or — <i>subject</i> was the unique subject and <i>subscribe_status</i> was FALSE (RTclient cannot stop subscribing to its unique subject), or — <i>subject</i> was not a valid subject • T_ERR_VAL_SAME — <i>subscribe_status</i> is the same as the current subscribe status • T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created

- Description** TpcSrvConnSubjectGetSubscribeLb and TpcSrvConnSubjectSetSubscribeLb are variants of TpcSrvConnSubjectGetSubscribe and TpcSrvConnSubjectSetSubscribe that include load balancing control.
- TpcSrvConnSubjectSetSubscribeLb sets whether or not this RTclient is subscribing to a subject, with or without load balancing. The subscribe status controls whether or not the RTclient receives messages published (sent) to that subject. If the subscribe status of *subject* is TRUE, then the RTclient receives messages with *subject* as their destination that are sent to RTserver. To determine if an RTclient is subscribing to a subject, including load balancing information, use TpcSrvConnSubjectGetSubscribeLb. If the load balancing status of *subject* is TRUE, then the RTclient is part of the load balancing calculation; otherwise the RTclient overrides load balancing and always receives the messages regardless of whether or not the message was published with load balancing.
- TpcSrvConnSubjectSetSubscribeLb may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TpcSrvConnSubjectSetSubscribeLb sets the subscribe status of the subject as if the RTclient were still connected to RTserver. See TpcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.
- Caution** TpcSrvConnSubjectSetSubscribeLb sends a SUBJECT_SET_SUBSCRIBE message to RTserver, but does not explicitly flush the message. See TpcSrvConnGetAutoFlushSize for more information on message buffering.
- See Also** TpcMonClientSubscribeSetWatch, TpcMonSubjectSubscribeSetWatch, TpcSrvConnSubjectGetSubscribeLb, TpcSrvConnSubjectSetSubscribe
- Examples** This example uses TpcSrvConnSubjectSetSubscribeLb to start subscribing to the subject `/system/thermal` and override load balancing so that this RTclient receives all messages published to the `thermal` subject regardless of whether or not they were published with load balancing:
- ```
if (!TpcSrvConnSubjectSetSubscribeLb(srv, "/system/thermal",
 TRUE, FALSE)) {
 return; /* error */
}
```



## TipcSrvConnSubjectSetUnique

---

|                      |                                                                                                                                                                                                                                                                                                                                                 |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvConnSubjectSetUnique — set the unique subject of the connection to RTserver                                                                                                                                                                                                                                                              |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvConnSubjectSetUnique(srv, unique_subject) T_IPC_SRV srv; T_STR unique_subject;</pre>                                                                                                                                                                                                                                         |
| <b>Arguments</b>     | <p><i>srv</i> — connection to RTserver</p> <p><i>unique_subject</i> — unique subject to set</p>                                                                                                                                                                                                                                                 |
| <b>Return Values</b> | TRUE if the unique subject was successfully set for the connection to RTserver, FALSE otherwise.                                                                                                                                                                                                                                                |
| <b>Diagnostics</b>   | <p>If TipcSrvConnSubjectSetUnique fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>unique_subject</i> was null</li> <li>• T_ERR_SRV_ALREADY_CONNECTED — a full or warm connection to RTserver was already established</li> </ul> |
| <b>Description</b>   | TipcSrvConnSubjectSetUnique sets the unique subject of the connection to RTserver.                                                                                                                                                                                                                                                              |
| <b>Caution</b>       | None                                                                                                                                                                                                                                                                                                                                            |
| <b>See Also</b>      | TipcSrvConnCreate, TipcSrvConnSubjectGetUnique                                                                                                                                                                                                                                                                                                  |
| <b>Examples</b>      | <p>This example sets the RTclient's unique subject:</p> <pre>if (!TipcSrvConnSubjectSetUnique(srv, "/trader")) {     return; /* error */ }</pre>                                                                                                                                                                                                |

## TipcSrvConnSubjectTraverseSubscribe

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvConnSubjectTraverseSubscribe — traverse the subjects that the RTclient is subscribing to                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Synopsis</b>      | <pre>T_PTR TipcSrvConnSubjectTraverseSubscribe(srv, func, arg) T_IPC_SRV srv; T_IPC_SRV_SUBJECT_TRAV_FUNC func; T_PTR arg;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>func</i> — function to call once for each subject</p> <p><i>arg</i> — user-defined argument to pass to <i>func</i></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Return Values</b> | The first non-null return value from <i>func</i> , or NULL if <i>func</i> always returns NULL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Diagnostics</b>   | <p>If TipcSrvConnSubjectTraverseSubscribe fails or traverses all the subjects an RTclient is subscribing to, it returns NULL and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null</li><li>• any error number from TutHashTraverse</li></ul>                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>   | <p>TipcSrvConnSubjectTraverseSubscribe traverses all the subjects an RTclient is subscribing to (the subjects with a subscribe status of TRUE) and calls <i>func</i> once for each subject. If <i>func</i> returns a non-null value, then TipcSrvConnSubjectTraverseSubscribe returns with that value.</p> <p>TipcSrvConnSubjectTraverseSubscribe may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnSubjectTraverseSubscribe traverses the subjects as if the RTclient were still connected to RTserver. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.</p> |
| <b>Caution</b>       | If TipcSrvConnSubjectTraverseSubscribe returns NULL, it may be because there was an error or simply because <i>func</i> always returned NULL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>See Also</b>      | TipcMonClientSubscribePoll, TipcMonSubjectSubscribePoll; see the <i>TIBCO SmartSockets Utilities</i> for information on TutHashTraverse.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

**Examples** This example prints the names of all subjects being subscribed to:

```
/* ===== */
/*..print_one_subject -- traverse one subject */
T_PTR print_one_subject(name, dummy, arg)
T_STR name;
T_STR dummy; /* not used */
T_PTR arg;
{
 TutOut("This process is subscribing to subject <%s>.\n", name);

 return NULL; /* continue traversal */
} /* print_one_subject */

/* ===== */
/*...code from calling function is below */

(void)TipcSrvConnSubjectTraverseSubscribe(srv, print_one_subject,
 NULL);
```

## TipcSrvConnTrafficGetBytesRecv8

**Name**      TipcConnTrafficGetBytesRecv8 — get the total number of bytes received on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

TipcConnTrafficGetBytesRecv — get the total number of bytes received on a connection

**Synopsis**      `T_BOOL TipcSrvConnTrafficGetBytesRecv8(srv, bytes_recv_return)`  
                 `T_IPC_SRV srv;`  
                 `T_INT8 *bytes_recv_return;`

**Arguments**      *srv* — connection handle to RTserver  
                 *bytes\_recv\_return* — returned number of bytes

**Return Values**      TRUE if the number of bytes was sucessfully retrieved, FALSE otherwise.

**Diagnostics**      If TipcSrvConnTrafficGetBytesRecv8 fails, it returns FALSE and sets the global SmartSockets error number to:

- `T_ERR_NULL_PTR` — *srv* or *bytes\_recv\_return* was null

**Description**      TipcSrvConnTrafficGetBytesRecv8 retrieves the total number of bytes received on a connection.

**See Also**      TipcSrvConnTrafficGetBytesSent8

**Examples**      This example retrieves the number of bytes received on a connection:

```
T_IPC_SRV srv;
T_INT8 bytes_recv;
.
.
.
if (!TipcSrvConnTrafficGetBytesRecv8(srv, &bytes_recv)) {
 return; /* error */
}
```

## TipcSrvConnTrafficGetBytesSent8

---

**Name**      TipcConnTrafficGetBytesSent8 — get the total number of bytes sent on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

TipcConnTrafficGetBytesSent — get the total number of bytes sent on a connection

**Synopsis**      `T_BOOL TipcSrvConnTrafficGetBytesSent8(srv, bytes_sent_return)`  
                  `T_IPC_SRV srv;`  
                  `T_INT8 *bytes_sent_return;`

**Arguments**    *srv* — connection handle to RTserver  
                  *bytes\_recv\_return* — returned number of bytes

**Return Values**    TRUE if the number of bytes was successfully retrieved, FALSE otherwise.

**Diagnostics**    If TipcSrvConnTrafficGetBytesSent8 fails, it returns FALSE and sets the global SmartSockets error number to:

- T\_ERR\_NULL\_PTR — *srv* or *bytes\_recv\_return* was null

**Description**    TipcSrvConnTrafficGetBytesSent8 retrieves the total number of bytes sent on a connection.

**See Also**        TipcSrvConnTrafficGetBytesRecv8

**Examples**        This example retrieves the number of bytes sent on a connection:

```
T_IPC_SRV srv;
T_INT8 bytes_sent;
.
.
.
if (!TipcSrvConnTrafficGetBytesSent8(srv, &bytes_sent)) {
 return; /* error */
}
```

## TipcSrvConnTrafficGetMsgsRecv8

**Name**      TipcConnTrafficGetMsgsRecv8 — get the total number of messages received on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

TipcConnTrafficGetMsgsRecv — get the total number of messages received on a connection

**Synopsis**      `T_BOOL TipcSrvConnTrafficGetMsgsRecv8(srv, msgs_recv_return)`  
                 `T_IPC_SRV srv;`  
                 `T_INT8 *msgs_recv_return;`

**Arguments**      *srv* — connection handle to RTserver  
                 *msgs\_recv\_return* — returned number of messages

**Return Values**      TRUE if the number of messages was sucessfully retrieved, FALSE otherwise.

**Diagnostics**      If TipcSrvConnTrafficGetMsgsRecv8 fails, it returns FALSE and sets the global SmartSockets error number to:

- `T_ERR_NULL_PTR` — *srv* or *msgs\_recv\_return* was null

**Description**      TipcSrvConnTrafficGetMsgsRecv8 retrieves the total number of messages received on a connection.

**See Also**      TipcSrvConnTrafficGetMsgsSent8

**Examples**      This example retrieves the number of messages received on a connection:

```
T_IPC_SRV srv;
T_INT8 msgs_recv;
.
.
.
if (!TipcSrvConnTrafficGetMsgsRecv8(srv, &msgs_recv)) {
 return; /* error */
}
```

## TipcSrvConnTrafficGetMsgsSent8

---

**Name**      `TipcConnTrafficGetMsgsSent8` — get the total number of messages sent on a connection



The following version of this function is deprecated in release 6.7 and later. The previous form supersedes it. We retain the older form only for backward source compatibility.

`TipcConnTrafficGetMsgsSent` — get the total number of messages sent on a connection

**Synopsis**

```
T_BOOL TipcSrvConnTrafficGetMsgsSent8(srv, msgs_sent_return)
T_IPC_SRV srv;
T_INT8 *msgs_sent_return;
```

**Arguments**      *srv* — connection handle to RTserver  
                   *msgs\_sent\_return* — returned number of messages

**Return Values**      TRUE if the number of messages was successfully retrieved, FALSE otherwise.

**Diagnostics**      If `TipcSrvConnTrafficGetMsgsSent8` fails, it returns FALSE and sets the global SmartSockets error number to:

- `T_ERR_NULL_PTR` — *srv* or *msgs\_sent\_return* was null

**Description**      `TipcSrvConnTrafficGetMsgsSent8` retrieves the total number of messages sent on a connection.

**See Also**        `TipcSrvConnTrafficGetMsgsRecv8`

**Examples**        This example retrieves the number of messages sent on a connection:

```
T_IPC_SRV srv;
T_INT8 msgs_sent;
.
.
.
if (!TipcSrvConnTrafficGetMsgsSent8(srv, &msgs_sent)) {
 return; /* error */
}
```

# TipcSrvConnUnlock

---

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvConnUnlock — release exclusive access to the connection                                                                                                                                                                                                                                                                                                                                                                                                       |
| Synopsis      | <pre>T_BOOL TipcSrvConnUnlock(srv) T_IPC_SRV srv;</pre>                                                                                                                                                                                                                                                                                                                                                                                                              |
| Arguments     | <i>srv</i> — connection handle to RTserver                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Return Values | TRUE if the connection to RTserver was successfully unlocked, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                       |
| Diagnostics   | <p>If TipcSrvConnUnlock fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> was null</li><li>• any error number from TutRwMutexUnlock</li></ul>                                                                                                                                                                                                                        |
| Description   | TipcSrvConnUnlock unlocks all of the internal synchronization objects within the connection to RTserver.                                                                                                                                                                                                                                                                                                                                                             |
| Caution       | Every call to TipcSrvConnLock must be balanced by a call to TipcSrvConnUnlock.                                                                                                                                                                                                                                                                                                                                                                                       |
| See Also      | TipcConnUnlock, TipcSrvConnLock; see the <i>TIBCO SmartSockets Utilities</i> for information on TutRwMutexUnlock.                                                                                                                                                                                                                                                                                                                                                    |
| Examples      | <p>This example locks and unlocks the connection to RTserver:</p> <pre>if (!TipcSrvConnLock(srv)) {     return; /* error */ }</pre> <p><i>/* At this point, several TipcSrvConn* functions could be called and there would be no risk of any other thread accessing the connection. The connection must be unlocked with TipcSrvConnUnlock when exclusive access is no longer needed. */</i></p> <pre>if (!TipcSrvConnUnlock(srv)) {     return; /* error */ }</pre> |



## TipcSrvConnWriteCbCreate

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvConnWriteCbCreate — create a write callback in the connection                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Synopsis</b>      | <pre>T_CB TipcSrvConnWriteCbCreate(srv, mt, func, arg) T_IPC_SRV srv; T_IPC_MT mt; T_IPC_CONN_WRITE_CB_FUNC func; T_CB_ARG arg;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>mt</i> — message type to create callback for (null means global callback)</p> <p><i>func</i> — callback function</p> <p><i>arg</i> — user-defined argument to pass to <i>func</i></p>                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Return Values</b> | New callback if successful, NULL otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Diagnostics</b>   | <p>If TipcSrvConnWriteCbCreate fails, it returns NULL and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null</li> <li>• T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created</li> <li>• any error number from TipcConnWriteCbCreate</li> </ul>                                                                                                                                                                                                                                                |
| <b>Description</b>   | <p>TipcSrvConnWriteCbCreate creates a write callback in the connection to RTserver by calling TipcConnWriteCbCreate. These callbacks are called when a message is sent on the connection to RTserver. See TipcConnWriteCbCreate for more information on write callbacks.</p> <p>TipcSrvConnWriteCbCreate may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnWriteCbCreate creates a write callback in the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.</p> |
| <b>Caution</b>       | The T_ENTRY declaration specifier is required in the definition of all callback functions as well as their prototypes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>See Also</b>      | TipcConnWriteCbCreate, TipcSrvConnWriteCbLookup                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

**Examples** This example creates a global write callback in the connection to RTserver that prints the name of the message type for each outgoing message:

```
/*===== */
/*..my_srv_write_cb -- print type name of outgoing message */
void T_ENTRY my_srv_write_cb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_WRITE_CB_DATA data;
T_CB_ARG arg; /* not used */
{
 T_IPC_MSG msg = data->msg;
 T_IPC_MT mt;
 T_STR name;

 if (!TipcMsgGetType(msg, &mt)) {
 return; /* error */
 }

 if (!TipcMtGetName(mt, &name)) {
 return; /* error */
 }

 /* print the message type name */
 TutOut("Outgoing message to RTserver with type %s.\n", name);
} /* my_srv_write_cb */

/*===== */
/*...code from main program is below */

if (TipcSrvConnWriteCbCreate(srv, NULL, /* NULL signifies global */
 my_srv_write_cb,
 NULL) == NULL) {
 return; /* error */
}
```

## TipcSrvConnWriteCbLookup

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvConnWriteCbLookup — look up a write callback in the connection                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Synopsis</b>      | <pre>T_CB TipcSrvConnWriteCbLookup(<i>srv</i>, <i>mt</i>, <i>func</i>, <i>arg</i>) T_IPC_SRV <i>srv</i>; T_IPC_MT <i>mt</i>; T_IPC_CONN_WRITE_CB_FUNC <i>func</i>; T_CB_ARG <i>arg</i>;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver to look up callback on</p> <p><i>mt</i> — message type to look up callback for (null means global callback)</p> <p><i>func</i> — callback function</p> <p><i>arg</i> — user-defined argument</p>                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Return Values</b> | New callback if successful, NULL otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Diagnostics</b>   | <p>If TipcSrvConnWriteCbLookup fails, it returns NULL and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null</li> <li>• T_ERR_SRV_NOT_CONNECTED — a connection to RTserver could not be created</li> <li>• any error number from TipcConnWriteCbLookup</li> </ul>                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b>   | <p>TipcSrvConnWriteCbLookup looks up a write callback with the desired message type, function, and argument in the connection to RTserver by calling TipcConnWriteCbLookup. These callbacks are called when a message is sent on the connection to RTserver. See TipcConnWriteCbCreate for more information on connection write callbacks.</p> <p>TipcSrvConnWriteCbLookup may attempt to automatically create a connection to RTserver. If RTclient has a warm connection to RTserver, TipcSrvConnWriteCbLookup looks up a write callback in the warm connection. See TipcSrvConnOpen for more information on automatic creation of connections and warm connections to RTserver.</p> |
| <b>Caution</b>       | None                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>See Also</b>      | TipcConnWriteCbLookup, TipcSrvConnWriteCbCreate; see the <i>TIBCO SmartSockets Utilities</i> for information on TutCbDestroy.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

**Examples** This example looks up and destroys a NUMERIC\_DATA write callback in the connection to RTserver:

```
T_CB cb;
T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_NUMERIC_DATA);
if (mt == NULL) {
 return; /* error */
}

cb = TipcSrvConnWriteCbLookup(srv, mt, my_numeric_write_cb, NULL);
if (cb == NULL) {
 return; /* error */
}

if (!TutCbDestroy(cb)) {
 return; /* error */
}
```

## Chapter 15 **TipcSrvMon\***

This chapter covers the TipcSrvMon API functions that handle multiple connections. For information on the TipcMon API functions that handle single server connections, see Chapter 5, TipcMon\*. This chapter covers these TipcSrvMon API functions:

- *TipcSrvMonClientBufferGet Watch*
- *TipcSrvMonClientBufferPoll*
- *TipcSrvMonClientBufferSet Watch*
- *TipcSrvMonClientCbPoll*
- *TipcSrvMonClientCongestionGet Watch*
- *TipcSrvMonClientCongestionSet Watch*
- *TipcSrvMonClientCpuPoll*
- *TipcSrvMonClientExtPoll*
- *TipcSrvMonClientGeneralPoll*
- *TipcSrvMonClientInfoPoll*
- *TipcSrvMonClientMsgRecvGet Watch*
- *TipcSrvMonClientMsgRecvSet Watch*
- *TipcSrvMonClientMsgSendGet Watch*
- *TipcSrvMonClientMsgSendSet Watch*
- *TipcSrvMonClientMsgTrafficPoll*
- *TipcSrvMonClientMsgTypeExPoll*
- *TipcSrvMonClientMsgTypePoll*
- *TipcSrvMonClientNamesGet Watch*
- *TipcSrvMonClientNamesNumPoll*
- *TipcSrvMonClientNamesPoll*
- *TipcSrvMonClientNamesSet Watch*
- *TipcSrvMonClientOptionPoll*
- *TipcSrvMonClientSubjectExPoll*

- *TpcSrvMonClientSubjectPoll*
- *TpcSrvMonClientSubscribeGet Watch*
- *TpcSrvMonClientSubscribeNumPoll*
- *TpcSrvMonClientSubscribePoll*
- *TpcSrvMonClientSubscribeSet Watch*
- *TpcSrvMonClientTimeGet Watch*
- *TpcSrvMonClientTimePoll*
- *TpcSrvMonClientTimeSet Watch*
- *TpcSrvMonClientVersionPoll*
- *TpcSrvMonExtDelete*
- *TpcSrvMonExtSetBinary*
- *TpcSrvMonExtSetBool*
- *TpcSrvMonExtSetBoolArray*
- *TpcSrvMonExtSetInt2*
- *TpcSrvMonExtSetInt2Array*
- *TpcSrvMonExtSetInt4*
- *TpcSrvMonExtSetInt4Array*
- *TpcSrvMonExtSetInt8*
- *TpcSrvMonExtSetInt8Array*
- *TpcSrvMonExtSetReal4*
- *TpcSrvMonExtSetReal4Array*
- *TpcSrvMonExtSetReal8*
- *TpcSrvMonExtSetReal8Array*
- *TpcSrvMonExtSetReal16*
- *TpcSrvMonExtSetReal16Array*
- *TpcSrvMonExtSetStr*

- *TipcSrvMonExtSetStrArray*
- *TipcSrvMonExtSetUtf8*
- *TipcSrvMonExtSetUtf8Array*
- *TipcSrvMonGetIdentStr*
- *TipcSrvMonPrint Watch*
- *TipcSrvMonProjectNamesGet Watch*
- *TipcSrvMonProjectNamesPoll*
- *TipcSrvMonProjectNamesSet Watch*
- *TipcSrvMonServerBufferPoll*
- *TipcSrvMonServerCongestionGet Watch*
- *TipcSrvMonServerCongestionSet Watch*
- *TipcSrvMonServerConnGet Watch*
- *TipcSrvMonServerConnPoll*
- *TipcSrvMonServerConnSet Watch*
- *TipcSrvMonServerCpuPoll*
- *TipcSrvMonServerGeneralPoll*
- *TipcSrvMonServerMaxClientLicensesGet Watch*
- *TipcSrvMonServerMaxClientLicensesSet Watch*
- *TipcSrvMonServerMsgTrafficExPoll*
- *TipcSrvMonServerMsgTrafficPoll*
- *TipcSrvMonServerNamesGet Watch*
- *TipcSrvMonServerNamesPoll*
- *TipcSrvMonServerNamesSet Watch*
- *TipcSrvMonServerOptionPoll*
- *TipcSrvMonServerRoutePoll*
- *TipcSrvMonServerStartTimePoll*

- *TpcSrvMonServerTimePoll*
- *TpcSrvMonServerVersionPoll*
- *TpcSrvMonSetIdentStr*
- *TpcSrvMonSubjectNamesGet Watch*
- *TpcSrvMonSubjectNamesPoll*
- *TpcSrvMonSubjectNamesSet Watch*
- *TpcSrvMonSubjectSubscribeGet Watch*
- *TpcSrvMonSubjectSubscribePoll*
- *TpcSrvMonSubjectSubscribeSet Watch*



## TipcSrvMonClientBufferGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientBufferGetWatch — determine if this RTclient is watching message-related buffer information in an RTclient                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientBufferGetWatch(srv, client_name, watch_status_return) T_IPC_SRV srv; T_STR client_name; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope)</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Return Values</b> | TRUE if the RTclient buffer information watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientBufferGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>T_ERR_NULL_PTR — <i>srv</i>, <i>client_name</i>, or <i>watch_status_return</i> was null</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>   | <p>TipcSrvMonClientBufferGetWatch determines whether this RTclient is watching message-related buffer information in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_BUFFER_STATUS message each time the message queue for the connection to RTserver changes in the watched RTclient.</p> <p>Each MON_CLIENT_BUFFER_STATUS message contains five fields:</p> <ul style="list-style-type: none"> <li>a STR field containing the name of the RTclient</li> <li>an INT4 field containing the number of queued incoming messages in the RTclient's connection to RTserver</li> <li>an INT4 field containing the number of bytes of data in the queued incoming messages in the RTclient's connection to RTserver</li> </ul> |

- an INT4 field containing the number of bytes of data in the read buffer in the RTclient's connection to RTserver
- an INT4 field containing the number of bytes of data in the write buffer in the RTclient's connection to RTserver

To set whether this RTclient is watching RTclient buffer information, use `TipcSrvMonClientBufferSetWatch`. To poll once for RTclient buffer information, use `TipcSrvMonClientBufferPoll`.

For an example of how `MON_CLIENT_BUFFER_STATUS` messages can be used, see the RTmon GDI Watch Client Buffer window.

**Caution** If `TipcSrvMonClientBufferGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

**See Also** `TipcSrvMonClientBufferPoll`, `TipcSrvMonClientBufferSetWatch`

**Examples** This example uses `TipcSrvMonClientBufferGetWatch` to get the status of watching buffer information in the primary RTclient:

```
T_BOOL watch_status;

if (!TipcSrvMonClientBufferGetWatch(srv, "primary",
&watch_status)) {
 return; /* error */
}
TutOut("This RTclient is %s RTclient buffer information in ",
 watch_status ? "watching" : "not watching");
TutOut("primary.\n");
```

## TipcSrvMonClientBufferPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientBufferPoll — poll once for RTclient message-related buffer information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientBufferPoll(srv, client_name) T_IPC_SRV srv; T_STR client_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll for buffer information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Return Values</b> | TRUE if the poll for RTclient buffer information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientBufferPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>client_name</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b>   | <p>TipcSrvMonClientBufferPoll polls for RTclient message-related buffer information by sending a MON_CLIENT_BUFFER_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_BUFFER_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_CLIENT_BUFFER_POLL_RESULT message contains five fields:</p> <ul style="list-style-type: none"> <li>• a STR field containing the name of the RTclient</li> <li>• an INT4 field containing the number of queued incoming messages in the RTclient's connection to RTserver</li> <li>• an INT4 field containing the number of bytes of data in the queued incoming messages in the RTclient's connection to RTserver</li> </ul> |

- an INT4 field containing the number of bytes of data in the read buffer in the RTclient's connection to RTserver
- an INT4 field containing the number of bytes of data in the write buffer in the RTclient's connection to RTserver

There are multiple responses only if *client\_name* is a wildcard or T\_IPC\_MON\_ALL.

To watch RTclient buffer information, use TipcSrvMonClientBufferSetWatch.

**Caution** TipcSrvMonClientBufferPoll sends a MON\_CLIENT\_BUFFER\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client\_name* does not exist; the poll is silently dropped.

**See Also** TipcSrvMonClientBufferGetWatch, TipcSrvMonClientBufferSetWatch, TipcSrvConnMsgWrite

**Examples** This example uses TipcSrvMonClientBufferPoll to poll for buffer information in all RTclients (including this one), creates a callback to process the incoming MON\_CLIENT\_BUFFER\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_buffer_poll_result -- process a MON_CLIENT_BUFFER_POLL_RESULT
message */
void T_ENTRY process_mon_client_buffer_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_INT4 msg_queue_count;
 T_INT4 msg_queue_byte_count;
 T_INT4 read_buffer_count;
 T_INT4 write_buffer_count;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }
}
```

```

/* get the fields from the message */
if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_INT4, &msg_queue_count,
 T_IPC_FT_INT4, &msg_queue_byte_count,
 T_IPC_FT_INT4, &read_buffer_count,
 T_IPC_FT_INT4, &write_buffer_count,
 NULL)) {
 return; /* error */
}

TutOut("Got CLIENT_BUFFER poll result.\n");
TutOut("RTclient name = %s\n", client_name);
TutOut("msg queue count = %d\n", msg_queue_count);
TutOut("msg queue byte count = %d\n", msg_queue_byte_count);
TutOut("read buffer count = %d\n", read_buffer_count);
TutOut("write buffer count = %d\n", write_buffer_count);
} /* process_mon_client_buffer_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientBufferPoll(srv, "/...")) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_BUFFER_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_buffer_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

# TipcSrvMonClientBufferSetWatch

---

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonClientBufferSetWatch — start or stop watching RTclient message-related buffer information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Synopsis      | <pre>T_BOOL TipcSrvMonClientBufferSetWatch(srv, client_name, watch_status) T_IPC_SRV srv; T_STR client_name; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to set watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope)</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Return Values | TRUE if the RTclient buffer information watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Diagnostics   | <p>If TipcSrvMonClientBufferSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> or <i>client_name</i> was null</li><li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li><li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Description   | <p>TipcSrvMonClientBufferSetWatch sets whether this RTclient is watching message-related buffer information in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The <i>srv</i> argument indicates the connection on which the watch is to be checked.</p> <p>TipcSrvMonClientBufferSetWatch saves the watch status and sends a MON_CLIENT_BUFFER_SET_WATCH message to the <i>client_name</i> RTclient. The watch status controls whether the RTclient receives a MON_CLIENT_BUFFER_STATUS message each time the message queue for the connection to RTserver changes in the watched RTclient. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.</p> |

Each `MON_CLIENT_BUFFER_STATUS` message contains five fields:

- a `STR` field containing the name of the `RTclient`
- an `INT4` field containing the number of queued incoming messages in the `RTclient`'s connection to `RTserver`
- an `INT4` field containing the number of bytes of data in the queued incoming messages in the `RTclient`'s connection to `RTserver`
- an `INT4` field containing the number of bytes of data in the read buffer in the `RTclient`'s connection to `RTserver`
- an `INT4` field containing the number of bytes of data in the write buffer in the `RTclient`'s connection to `RTserver`

In addition to sending a `MON_CLIENT_BUFFER_STATUS` message each time the message queue changes, the watched `RTclient` also sends an initial `MON_CLIENT_BUFFER_STATUS` message when the watch status is set to `TRUE`, so that the `RTclient` does not have to wait for the next message queue change to receive the first `MON_CLIENT_BUFFER_STATUS` message.

To determine if this `RTclient` is watching `RTclient` buffer information, use `TipcSrvMonClientBufferGetWatch`. To poll once for `RTclient` buffer information, use `TipcSrvMonClientBufferPoll`.

For an example of how `MON_CLIENT_BUFFER_STATUS` messages can be used, see the `RTmon GDI Watch Client Buffer` window.

**Caution** `TipcSrvMonClientBufferSetWatch` sends a `MON_CLIENT_BUFFER_SET_WATCH` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no initial status message if the `RTclient` named *client\_name* does not exist.

Buffer changes caused by `MON_CLIENT_BUFFER_STATUS`, `MON_CLIENT_MSG_RECV_STATUS`, and `MON_CLIENT_MSG_SEND_STATUS` message types cannot be watched and are silently ignored by the internal monitoring callbacks that generate outgoing `MON_CLIENT_BUFFER_STATUS` messages. The callbacks ignore buffer changes due to these message types to prevent infinite loops.

**See Also** `TipcSrvMonClientBufferGetWatch`, `TipcSrvMonClientBufferPoll`

**Examples** This example uses TpcSrvMonClientBufferSetWatch to start watching buffer information in the primary RTclient and creates a callback to process the incoming MON\_CLIENT\_BUFFER\_STATUS messages:

```
/* ===== */
/*..process_mon_client_buffer_status -- process a MON_CLIENT_BUFFER_STATUS message */

void T_ENTRY process_mon_client_buffer_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_INT4 msg_queue_count;
 T_INT4 msg_queue_byte_count;
 T_INT4 read_buffer_count;
 T_INT4 write_buffer_count;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_INT4, &msg_queue_count,
 T_IPC_FT_INT4, &msg_queue_byte_count,
 T_IPC_FT_INT4, &read_buffer_count,
 T_IPC_FT_INT4, &write_buffer_count,
 NULL)) {
 return; /* error */
 }

 TutOut("Got CLIENT_BUFFER status.\n");
 TutOut("RTclient name = %s\n", client_name);
 TutOut("msg queue count = %d\n", msg_queue_count);
 TutOut("msg queue byte count = %d\n", msg_queue_byte_count);
 TutOut("read buffer count = %d\n", read_buffer_count);
 TutOut("write buffer count = %d\n", write_buffer_count);
} /* process_mon_client_buffer_status */
```



```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonClientBufferSetWatch(srv, "primary", TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_BUFFER_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_buffer_status,
 NULL)
 == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages.*/

```

## TipcSrvMonClientCbPoll

---

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonClientCbPoll — poll once for RTclient callback information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Synopsis      | <pre>T_BOOL TipcSrvMonClientCbPoll(<i>srv</i>, <i>client_name</i>) T_IPC_SRV <i>srv</i>; T_STR <i>client_name</i>;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll for callback information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Return Values | TRUE if the poll for RTclient callback information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Diagnostics   | <p>If TipcSrvMonClientCbPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>T_ERR_NULL_PTR — <i>srv</i> or <i>client_name</i> was null</li><li>any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Description   | <p>TipcSrvMonClientCbPoll polls for RTclient callback information by sending a MON_CLIENT_CB_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_CB_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_CLIENT_CB_POLL_RESULT message contains nine fields:</p> <ul style="list-style-type: none"><li>a STR field containing the name of the RTclient</li><li>an INT4 field containing the number of global read callbacks of the polled RTclient connection</li><li>an INT4 field containing the number of global write callbacks of the polled RTclient connection</li><li>an INT4 field containing the number of global process callbacks of the polled RTclient connection</li><li>an INT4 field containing the number of global queue callbacks of the polled RTclient connection</li></ul> |

- an INT4 field containing the number of default callbacks of the polled RTclient connection
- an INT4 field containing the number of error callbacks of the polled RTclient connection
- an INT4 field containing the number of server create callbacks of the polled RTclient connection
- an INT4 field containing the number of server destroy callbacks of the polled RTclient connection

There are multiple responses only if *client\_name* is a wildcard or T\_IPC\_MON\_ALL.

**Caution** TipcSrvMonClientCbPoll sends a MON\_CLIENT\_CB\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client\_name* does not exist; the poll is silently dropped.

RTclient callback information can only be polled, not watched.

There is no way to poll for RTserver callback information.

**See Also** TipcSrvMonClientMsgTrafficPoll, TipcSrvConnMsgWrite, TipcSrvConnGetAutoFlushSize

**Examples** This example uses `TipcSrvMonClientCbPoll` to poll for callback information in all RTclients, including this one, creates a callback to process the incoming `MON_CLIENT_CB_POLL_RESULT` messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_cb_poll_result -- process a MON_CLIENT_CB_POLL_RESULT message
*/
void T_ENTRY process_mon_client_cb_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_INT4 num_global_read_cb;
 T_INT4 num_global_write_cb;
 T_INT4 num_global_process_cb;
 T_INT4 num_global_queue_cb;
 T_INT4 num_default_cb;
 T_INT4 num_error_cb;
 T_INT4 num_server_create_cb;
 T_INT4 num_server_destroy_cb;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_INT4, &num_global_read_cb,
 T_IPC_FT_INT4, &num_global_write_cb,
 T_IPC_FT_INT4, &num_global_process_cb,
 T_IPC_FT_INT4, &num_global_queue_cb,
 T_IPC_FT_INT4, &num_default_cb,
 T_IPC_FT_INT4, &num_error_cb,
 T_IPC_FT_INT4, &num_server_create_cb,
 T_IPC_FT_INT4, &num_server_destroy_cb,
 NULL)) {

return; /* error */
 }

 TutOut("Got CLIENT_CB poll result.\n");
 TutOut("RTclient name = %s\n", client_name);
 TutOut("num global read cb = %d\n", num_global_read_cb);
 TutOut("num global write cb = %d\n", num_global_write_cb);
 TutOut("num global process cb = %d\n", num_global_process_cb);
 TutOut("num global queue cb = %d\n", num_global_queue_cb);
 TutOut("num default cb = %d\n", num_default_cb);
 TutOut("num error cb = %d\n", num_error_cb);
 TutOut("num server create cb = %d\n", num_server_create_cb);
 TutOut("num server destroy cb = %d\n", num_server_destroy_cb);
} /* process_mon_client_cb_poll_result */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientCbPoll(srv, "/...")) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_CB_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_cb_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonClientCongestionGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientCongestionGetWatch — determine if the RTclient is watching a client's write buffer for congestion                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientCongestionGetWatch(srv, client_name, watch_status_return) T_IPC_SRV srv; T_STR client_name; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of the RTclient for which a watch is being checked</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Return Values</b> | TRUE if the watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientCongestionGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>client_name</i> or <i>watch_status_return</i> was null</li></ul>                                                                                                                                                                                                                                                                                                                                                        |
| <b>Description</b>   | <p>TipcSrvMonClientCongestionGetWatch determines whether the RTclient is watching for congestion in the <i>client_name</i> RTclient write buffer. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The <i>srv</i> argument indicates the connection on which the watch is to be checked.</p> <p>The <i>watch_status_return</i> argument is set to TRUE if the watch is currently active, FALSE if not.</p> |
| <b>Caution</b>       | <p>TipcSrvMonClientCongestionGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i>.</p> <p>RTclient write buffer congestion can only be watched, not polled.</p>                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>See Also</b>      | TipcSrvMonClientBufferSetWatch, TipcSrvMonClientCongestionSetWatch, TipcSrvMonServerCongestionGetWatch, TipcSrvMonServerCongestionSetWatch                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

**Examples** This example uses TipcSrvMonClientCongestionGetWatch to determine the status of a buffer congestion watch in the primary RTclient:

```
T_BOOL watch_status;

if (!TipcSrvMonClientCongestionGetWatch(srv, "primary",
 &watch_status)) {
 return; /* error */
}

TutOut("This RTclient is %s RTclient buffer congestion in ",
 watch_status ? "watching" : "not watching");
TutOut("primary.\n");
```

## TipcSrvMonClientCongestionSetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientCongestionSetWatch — start or stop watching for congestion in an RTclient’s write buffer                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientCongestionSetWatch(srv, client_name, high_water, low_water, watch_status) T_IPC_SRV srv; T_STR client_name; T_INT4 high_water; T_INT4 low_water; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of the RTclient to set watch status for</p> <p><i>high_water</i> — the number of pending bytes that must be in the buffer for it to be considered congested</p> <p><i>low_water</i> — the quantity that the number of pending bytes must fall below to end buffer congestion</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, or FALSE to stop watching)</p>                                                                                                                                                                                                                                                               |
| <b>Return Values</b> | TRUE if the watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientCongestionSetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>client_name</i> was null</li><li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE, or the value of <i>low_water</i> was less than 0 or greater than the value of <i>high_water</i></li><li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li></ul>                                                                                                                                                                                                                                           |
| <b>Description</b>   | <p>TipcSrvMonClientCongestionSetWatch starts or stops the watch of the write buffer in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The <i>srv</i> argument indicates the connection on which the watch is to be set.</p> <p>TipcSrvMonClientCongestionSetWatch saves the watch status and sends a MON_CLIENT_CONGESTION_SET_WATCH message to the RTserver. The watch status controls whether the RTclient receives a MON_CLIENT_CONGESTION_STATUS message when the number of bytes in</p> |



the watched RTclient's write buffer passes the *high\_water* or *low\_water* threshold. When the write buffer reaches the value of *high\_water*, the RTserver sends a `MON_CLIENT_CONGESTION_STATUS` message. The next `MON_CLIENT_CONGESTION_STATUS` message is sent when the number of bytes in the write buffer decreases to the *low\_water* number.

If *watch\_status* is `FALSE`, the values in *high\_water* and *low\_water* are ignored.

The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.

Each `MON_CLIENT_CONGESTION_STATUS` message contains five fields:

- an INT4 field containing the current size of the write buffer
- an INT4 field containing the threshold value passed (the *high\_water* or *low\_water* number)
- a STR field containing the *client\_name*
- a BOOL field containing which threshold was passed (`TRUE` for the *high\_water* threshold, `FALSE` if the *low\_water* threshold was passed)

Once the number of bytes in the write buffer has reached the *high\_water* number, RTserver does not send another `MON_CLIENT_CONGESTION_STATUS` message until number of pending bytes in the buffer decreases to the *low\_water* number.

**Caution** TipcSrvMonClientCongestionSetWatch sends a `MON_CLIENT_CONGESTION_SET_WATCH` message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

RTclient write buffer congestion can only be watched, not polled.

**See Also** TipcSrvMonClientBufferGetWatch, TipcSrvMonClientCongestionGetWatch, TipcSrvMonServerCongestionSetWatch, TipcSrvMonServerCongestionGetWatch

**Examples** This example uses TpcSrvMonClientCongestionSetWatch to start watching buffer congestion in the *primary\_rtie* RTclient with a *high\_water* threshold of 8 million bytes and a *low\_water* threshold of 2 million bytes, and creates a callback to process the incoming MON\_CLIENT\_CONGESTION\_STATUS messages:

```
/* ===== */
/*..process_mon_client_congestion_status – process a
MON_CLIENT_CONGESTION_STATUS message */

void process_mon_client_congestion_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_INT4 size;
 T_INT4 threshold;
 T_STR client_name;
 T_BOOL high_water;

 /* must set current field first */
 if (!TpcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TpcMsgRead(data->msg,
 T_IPC_FT_INT4, &size,
 T_IPC_FT_INT4, &threshold,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_BOOL, &high_water,
 NULL)) {
 return; /* error */
 }

 TutOut("Got CLIENT_CONGESTION status.\n");
 TutOut("Crossed %s water threshold.\n", high_water ? "high" :
"low");
 TutOut("RTclient name = %s\n", client_name);
 TutOut("buffer size = %d\n", size);
 TutOut("threshold = %d\n", threshold);
} /* process_mon_client_congestion_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientCongestionSetWatch (srv, "primary_rtie",
8000000,
2000000, TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_CONGESTION_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
process_mon_client_congestion_status,
NULL) == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```

# TipcSrvMonClientCpuPoll

---

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonClientCpuPoll — poll for the percentage of CPU time used by an RTclient                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Synopsis      | <pre>T_BOOL TipcSrvMonClientCpuPoll(srv, client_name) T_IPC_SRV srv; T_STR client_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll CPU usage for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Return Values | TRUE if the poll for CPU usage was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Diagnostics   | <p>If TipcSrvMonClientCpuPoll fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>T_ERR_NULL_PTR — <i>client_name</i> was null</li><li>any error number from TipcSrvMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Description   | <p>TipcSrvMonClientCpuPoll polls for the percentage of CPU time used by an RTclient by sending a MON_CLIENT_CPU_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The <i>srv</i> argument indicates the connection on which the poll is to be sent.</p> <p>The polled RTclient responds by sending back a MON_CLIENT_CPU_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. Each MON_CLIENT_CPU_POLL_RESULT contains two fields:</p> <ul style="list-style-type: none"><li>a STR field containing the name of the RTclient</li><li>a REAL4 field containing the percentage of CPU time used (between 0.0 and 100.0)</li></ul> <p>There are multiple responses only if <i>client_name</i> is a wildcard or T_IPC_MON_ALL.</p> <p>If the RTclient cannot query the operating system, the value reported in the CPU percentage used field is a negative number.</p> |

**Caution** There is no response message if the RTclient named *client\_name* does not exist; the poll is silently dropped.

TipcSrvMonClientCpuPoll sends a MON\_CLIENT\_CPU\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

RTclient CPU usage information can only be polled, not watched.

**See Also** TipcSrvMonServerCpuPoll

**Examples** This example uses TipcSrvMonClientCpuPoll to poll for information in all RTclients, including the RTclient initiating the poll, creates a callback to process the incoming MON\_CLIENT\_CPU\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_cpu_poll_result - process a
MON_CLIENT_CPU_POLL_RESULT message */

void process_mon_client_cpu_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_REAL4 cpu_utilization;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_REAL4, &cpu_utilization,
 NULL)) {
 return; /* error */
 }

 TutOut("Got CLIENT_CPU poll result.\n");
 TutOut("RTclient name = %s\n", client_name);
 TutOut("CPU utilization percentage = %f\n", cpu_utilization);
} /* process_mon_client_cpu_poll_result */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientCpuPoll(srv, "...")) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_CPU_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_cpu_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming messages) that arrive within the
next 10 seconds */

if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonClientExtPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientExtPoll — poll for extension data of an RTclient                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientExtPoll(<i>srv</i>, <i>client_name</i>) T_IPC_SRV <i>srv</i>; T_STR <i>client_name</i>;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>client_name</i> — name of the RTclient to retrieve extension data from. The value of <i>client_name</i> can be:</p> <ul style="list-style-type: none"> <li>• the unique subject name of an RTclient</li> <li>• a subject specified with wildcard characters to match multiple RTclients; for example, <code>/stocks/auto/...</code> or <code>/stocksla*/...</code></li> <li>• <code>T_IPC_MON_ALL</code> which specifies to poll all RTclients in the project that match the value of the <code>Monitor_Scope</code> option</li> </ul> |
| <b>Return Values</b> | TRUE if the poll for RTclient extension data was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientExtPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• <code>T_ERR_NULL_PTR</code> — <i>client_name</i> was null</li> <li>• any error number from TipcSrvConnMsgWriter</li> </ul>                                                                                                                                                                                                                                                                                                             |
| <b>Description</b>   | <p>TipcSrvMonClientExtPoll requests extension data from one or more RTclients by sending a message of type <code>MON_CLIENT_EXT_POLL_CALL</code> to those RTclients. Extension data is defined as any information generated by an RTclient which another RTclient wants to monitor. The extension data is not generated by SmartSockets.</p> <p>One or more RTclients respond to a <code>MON_CLIENT_EXT_POLL_CALL</code> message by returning their data in a message of type <code>MON_CLIENT_EXT_POLL_RESULT</code>.</p>                                                                         |
| <b>See Also</b>      | Use the necessary TipcSrvMonExt* APIs to define extension data in each RTclient that is requested by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

**Examples** In this example, RTclient1 copies the value of its balance field (100.0) into the field called account\_balance in its MON\_CLIENT\_EXT\_POLL\_RESULT message. RTclient2 requests the account information from RTclient 1 with TipcMonClientExtPoll and prints the account balance:

**RTclient 1:**

```
.
.
.
.
 T_REAL4 balance=100.0;
 TipcSrvMonExtSetReal4(srv1,"account_balance",balance);
 TipcSrvConnMainLoop(srv1,T_TIMEOUT_FOREVER);
 TipcSrvMonExtDelete(srv1,"account_balance");
.
.
.
```

**RTclient 2:**

```
.
.
.
void T_ENTRY ext_poll_result_cb
(
 T_IPC_CONN conn,
 T_IPC_CONN_MSG_CB_DATA data,
 T_PTR arg
)
{
 T_STR client;
 T_REAL4 account_balance = 0;
 T_INT4 num_named_fields = 0;
 T_BOOL has_account_balance;

 TipcMsgSetCurrent(data->msg,0);

 TipcMsgNextStr(data->msg,&client);
 TipcMsgNextInt4(data->msg,&num_named_fields);
```



```

/* Get "account_balance" extension data from RTclient1 */

has_account_balance = TipcMsgGetNamedReal4(data->msg,
 "account_balance",
 &account_balance);

if (T_TRUE == has_account_balance) {
 TutOut("Client %s has an account balance of %f dollars.\n",
 client,
 account_balance);
}
}
.
.
.
TipcSrvConnProcessCbCreate(srv2,
 TipcMtLookupByNum(T_MT_MON_CLIENT_EXT_POLL_RESULT),
 ext_poll_result_cb, T_NULL);
TipcSrvMonClientExtPoll(srv2, T_IPC_MON_ALL);
TipcSrvConnMainLoop(srv2, T_TIMEOUT_FOREVER);
.
.
.

```

# TipcSrvMonClientGeneralPoll

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonClientGeneralPoll — poll once for RTclient general information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Synopsis      | <pre>T_BOOL TipcSrvMonClientGeneralPoll(srv, client_name) T_IPC_SRV srv; T_STR client_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll for general information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Return Values | TRUE if the poll for RTclient general information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Diagnostics   | <p>If TipcSrvMonClientGeneralPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> or <i>client_name</i> was null</li><li>• any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Description   | <p>TipcSrvMonClientGeneralPoll polls for RTclient general information by sending a MON_CLIENT_GENERAL_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_GENERAL_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_CLIENT_GENERAL_POLL_RESULT message contains sixteen fields:</p> <ul style="list-style-type: none"><li>• a STR field containing the name of the RTclient</li><li>• a STR field containing the process monitoring identifier (defaults to a major type of license, such as Development_RTmon)</li><li>• a STR field containing the node name the RTclient is running on</li><li>• a STR field containing the user name the RTclient is running under</li><li>• an INT4 field containing the PID of the RTclient</li><li>• a STR field containing the project the RTclient is a part of</li></ul> |

- a STR field containing the name of the RTserver the RTclient is connected to
- a STR field containing the name of the SmartSockets architecture (such as `sun4_solaris`)
- an INT4 field containing the current operating system `sbrk` value (the edge of the RTclient's virtual memory address space)
- an INT4 field containing the amount of change in the current operating system `sbrk` value (the edge of the process's virtual memory address space) since the RTclient started (actually, since the process's first call to any SmartSockets function)
- an INT2 field containing the integer number format of the RTclient (really of type `T_INT_FORMAT`, but stored in an INT2 field to save space)
- an INT2 field containing the real number format of the RTclient (really of type `T_REAL_FORMAT`, but stored in an INT2 field to save space)
- a STR field containing the fully expanded logical connection name of the connection to RTserver
- an INT4\_ARRAY field containing the counted licenses that the RTclient has checked out (really an array of type `T_LICENSE`, but stored in an INT4\_ARRAY field for portability)
- a STR\_ARRAY field containing the extra licenses that the RTclient has checked out (such as `snmp`)
- a STR\_ARRAY field containing the names of the subjects that the RTclient is subscribing to

There are multiple responses only if *client\_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTserver general information, use `TipcSrvMonServerGeneralPoll`.

For an example of how `MON_CLIENT_GENERAL_POLL_RESULT` messages can be used, see the RTmon GDI Client Information window.

**Caution** `TipcSrvMonClientGeneralPoll` sends a `MON_CLIENT_GENERAL_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTclient named *client\_name* does not exist; the poll is silently dropped.

RTclient general information can only be polled, not watched.

**See Also** `TipcSrvMonServerGeneralPoll`, `TipcSrvConnMsgWrite`

**Examples** This example uses TIPC\_SrvMonClientGeneralPoll to poll for general information in all RTclients (including this one), creates a callback to process the incoming MON\_CLIENT\_GENERAL\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_general_poll_result -- process a
MON_CLIENT_GENERAL_POLL_RESULT message */
void T_ENTRY process_mon_client_general_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_STR ident;
 T_STR node_name;
 T_STR user_name;
 T_INT4 pid;
 T_STR project;
 T_STR server_name;
 T_STR arch;
 T_INT4 current_sbrk;
 T_INT4 delta_sbrk;
 T_INT2 int_format;
 T_INT2 real_format;
 T_STR logical_conn_name;
 T_INT4 *counted_lic;
 T_INT4 num_counted_lic;
 T_STR *extra_lic;
 T_INT4 num_extra_lic;
 T_STR *subject_subscribe;
 T_INT4 num_subject_subscribe;
 T_INT4 counter; /* loop counter */

 /* must set current field first */
 if (!TIPCMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }
}
```

```

/* get the fields from the message */
if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_STR, &ident,
 T_IPC_FT_STR, &node_name,
 T_IPC_FT_STR, &user_name,
 T_IPC_FT_INT4, &pid,
 T_IPC_FT_STR, &project,
 T_IPC_FT_STR, &server_name,
 T_IPC_FT_STR, &arch,
 T_IPC_FT_INT4, &current_sbrk,
 T_IPC_FT_INT4, &delta_sbrk,
 T_IPC_FT_INT2, &int_format,
 T_IPC_FT_INT2, &real_format,
 T_IPC_FT_STR, &logical_conn_name,
 T_IPC_FT_INT4_ARRAY, &counted_lic,
 &num_counted_lic,
 T_IPC_FT_STR_ARRAY, &extra_lic,
 &num_extra_lic,
 T_IPC_FT_STR_ARRAY, &subject_subscribe,
 &num_subject_subscribe,
 NULL)) {
 return; /* error */
}

TutOut("Got CLIENT_GENERAL poll result.\n");
TutOut("RTclient name = %s\n", client_name);
TutOut("ident = %s\n", ident);
TutOut("node name = %s\n", node_name);
TutOut("user name = %s\n", user_name);
TutOut("pid = %d\n", pid);
TutOut("project = %s\n", project);
TutOut("RTserver name = %s\n", server_name);
TutOut("architecture = %s\n", arch);
TutOut("current sbrk value = %d\n", current_sbrk);
TutOut("delta sbrk value = %d\n", delta_sbrk);
TutOut("int format = %d\n", int_format);
TutOut("real format = %d\n", real_format);
TutOut("logical conn name = %s\n", logical_conn_name);
TutOut("counted licenses\n");
for (counter = 0; counter < num_counted_lic; counter++) {
 TutOut(" %d\n", counted_lic[counter]);
}
TutOut("extra licenses\n");
for (counter = 0; counter < num_extra_lic; counter++) {
 TutOut(" %s\n", extra_lic[counter]);
}
TutOut("subjects being subscribed to\n");
for (counter = 0; counter < num_subject_subscribe; counter++) {
 TutOut(" %s\n", subject_subscribe[counter]);
}
} /* process_mon_client_general_poll_result */

```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientGeneralPoll(srv, "/...")) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_GENERAL_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
process_mon_client_general_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonClientInfoPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientInfoPoll — poll for RTclient information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientInfoPoll(srv, client_name) T_IPC_SRV srv; T_STR client_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Return Values</b> | TRUE if the poll was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientInfoPoll fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>client_name</i> was null</li> <li>• any error number from TipcSrvMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Description</b>   | <p>TipcSrvMonClientInfoPoll polls for RTclient information by sending a MON_CLIENT_INFO_POLL_CALL message to the RTserver. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The <i>srv</i> argument indicates the connection on which the poll is to be sent.</p> <p>The RTserver responds by sending back a MON_CLIENT_INFO_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. Each MON_CLIENT_INFO_POLL_RESULT message contains 15 fields:</p> <ul style="list-style-type: none"> <li>• a STR field containing the name of the RTclient</li> <li>• a STR field containing the process monitoring identifier (defaults to a major type of license, such as Development_RTmon)</li> <li>• a STR field containing the node name the RTclient is running on</li> <li>• a STR field containing the user name the RTclient is running under</li> <li>• an INT4 field containing the PID of the RTclient</li> <li>• a STR field containing the project the RTclient is a part of</li> <li>• a STR field containing the name of the RTserver the RTclient is connected to</li> </ul> |

- a STR field containing the name of the platform (such as `sun4_solaris`)
- an INT4 field containing the current operating system `sbrk` value (the edge of the RTclient's virtual memory address space)
- an INT4 field containing the amount of change in the current operating system `sbrk` value (the edge of the process's virtual memory address space) since the RTclient started (actually, since the process's first call to any SmartSockets function)
- an INT2 field containing the integer number format of the RTclient (really of type `T_INT_FORMAT`, but stored in an INT2 field to save space)
- an INT2 field containing the real number format of the RTclient (really of type `T_REAL_FORMAT`, but stored in an INT2 field to save space)
- a STR field containing the fully expanded logical connection name of the connection to RTserver
- and INT4 field containing the number of subjects the RTclient is subscribed to
- a REAL4 field containing the percentage of CPU time used (between 0.0 and 100.0) for the RTclient

There are multiple responses only if *client\_name* is a wildcard or `T_IPC_MON_ALL`.

If the RTclient cannot query the operating system, the value reported in the CPU percentage used field is a negative number.

**Caution** There is no response message if the RTclient named *client\_name* does not exist; the poll is silently dropped.

`TpcSrvMonClientInfoPoll` sends a `MON_CLIENT_INFO_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TpcSrvConnGetAutoFlushSize` for more information on message buffering.

RTclient information can only be polled, not watched.

**See Also** `TpcSrvMonClientGeneralPoll`



**Examples** This example uses TipcSrvMonClientInfoPoll to

- poll for information in all RTclients, including the RTclient initiating the poll
- create a callback to process incoming MON\_CLIENT\_INFO\_POLL\_RESULT messages
- wait up to 10 seconds for the poll results

```
/* ===== */
/*..process_mon_client_info_poll_result - process a
MON_CLIENT_INFO_POLL_RESULT message */

void process_mon_client_info_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_STR ident;
 T_STR node_name;
 T_STR user_name;
 T_INT4 pid;
 T_STR project;
 T_STR server_name;
 T_STR arch;
 T_INT4 current_sbrk;
 T_INT4 delta_sbrk;
 T_INT2 int_format;
 T_INT2 real_format;
 T_STR logical_conn_name;
 T_INT4 num_subject_subscribe;
 T_REAL4 cpu_utilization;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_STR, &ident,
 T_IPC_FT_STR, &node_name,
 T_IPC_FT_STR, &user_name,
 T_IPC_FT_INT4, &pid,
 T_IPC_FT_STR, &project,
 T_IPC_FT_STR, &server_name,
 T_IPC_FT_STR, &arch,
 T_IPC_FT_INT4, &current_sbrk,
 T_IPC_FT_INT4, &delta_sbrk,
```

```

 T_IPC_FT_INT2, &int_format,
 T_IPC_FT_INT2, &real_format,
 T_IPC_FT_STR, &logical_conn_name,
 T_IPC_FT_INT4, &num_subject_subscribe,
 T_IPC_FT_REAL4, &cpu_utilization,
 NULL)) {
 return; /* error */
}

TutOut("Got CLIENT_INFO poll result.\n");
TutOut("RTclient name = %s\n", client_name);
TutOut("ident = %s\n", ident);
TutOut("node name = %s\n", node_name);
TutOut("user name = %s\n", user_name);
TutOut("pid = %d\n", pid);
TutOut("project = %s\n", project);
TutOut("RTserver name = %s\n", server_name);
TutOut("architecture = %s\n", arch);
TutOut("current sbrk value = %d\n", current_sbrk);
TutOut("delta sbrk value = %d\n", delta_sbrk);
TutOut("int format = %d\n", int_format);
TutOut("real format = %d\n", real_format);
TutOut("logical conn name = %s\n", logical_conn_name);
TutOut("number of subject subscribes = %d\n",
num_subject_subscribe);
TutOut("CPU utilization percentage = %f\n", cpu_utilization);
} /* process_mon_client_info_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientInfoPoll(srv, "/...")) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_INFO_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_info_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming messages) that arrive within the
next 10 seconds */

if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonClientMsgRecvGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientMsgRecvGetWatch — determine if this RTclient is watching received messages in an RTclient                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientMsgRecvGetWatch(srv, client_name, msg_type_name, watch_status_return) T_IPC_SRV srv; T_STR client_name; T_STR msg_type_name; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope)</p> <p><i>msg_type_name</i> — name of message type to get watch status for (use T_IPC_MON_ALL to indicate a watch of all message types)</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Return Values</b> | TRUE if the RTclient received messages watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientMsgRecvGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>T_ERR_NULL_PTR — <i>srv</i>, <i>client_name</i>, <i>msg_type_name</i>, or <i>watch_status_return</i> was null</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Description</b>   | <p>TipcSrvMonClientMsgRecvGetWatch determines whether this RTclient is watching received messages in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_MSG_RECV_STATUS message each time a received message is inserted into or deleted from the message queue (for the connection to RTserver) in the watched RTclient. If <i>msg_type_name</i> is T_IPC_MON_ALL, then all message types in the watched RTclient are watched.</p> <p>Each MON_CLIENT_MSG_RECV_STATUS message contains six fields:</p> <ul style="list-style-type: none"> <li>a STR field containing the name of the RTclient</li> <li>a MSG field containing the received message</li> </ul> |

- an INT4 field containing whether the message was just inserted into or deleted from the connection's message queue (really of type T\_BOOL); a value of TRUE means the message was inserted, and a value of FALSE means the message was deleted
- an INT4 field containing the position in the queue where the received message was just inserted or deleted (a value of 0 means the front of the queue)
- a BINARY field containing a unique 8-byte identifier for the received message; this field is useful for processes that need to accurately track the entire contents of the connection's message queue, such as the RTmon GDI
- a BINARY field containing an array of unique 8-byte identifiers for all messages in the connection's message queue; this field is useful for processes that need to accurately track the entire contents of the connection's message queue, such as the RTmon GDI

To set whether this RTclient is watching RTclient received messages, use `TipcSrvMonClientMsgRecvSetWatch`.

For an example of how `MON_CLIENT_MSG_RECV_STATUS` messages can be used, see the RTmon GDI Watch Messages Received window.

**Caution** If `TipcSrvMonClientMsgRecvGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

RTclient received messages can only be watched, not polled.

**See Also** `TipcSrvMonClientMsgRecvSetWatch`

**Examples** This example uses `TipcSrvMonClientMsgRecvGetWatch` to get the status of watching all received messages in the primary RTclient:

```
T_BOOL watch_status;

if (!TipcSrvMonClientMsgRecvGetWatch(srv, "primary",
T_IPC_MON_ALL,
 &watch_status)) {
 return; /* error */
}
TutOut("This process is %s all messages received in primary.\n",
 watch_status ? "watching" : "not watching");
```

## TipcSrvMonClientMsgRecvSetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientMsgRecvSetWatch — start or stop watching RTclient received messages                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientMsgRecvSetWatch(srv, client_name, msg_type_name, watch_status) T_IPC_SRV srv; T_STR client_name; T_STR msg_type_name; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to set watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope)</p> <p><i>msg_type_name</i> — name of message type to set watch status for (use T_IPC_MON_ALL to indicate a watch of all message types)</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                    |
| <b>Return Values</b> | TRUE if the RTclient received watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientMsgRecvSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i>, <i>client_name</i>, or <i>msg_type_name</i> was null</li> <li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li> <li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li> </ul>                                                                                                                                                                                                                                                                                                                                        |
| <b>Description</b>   | <p>TipcSrvMonClientMsgRecvSetWatch sets whether this RTclient is watching received messages in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option.</p> <p>TipcSrvMonClientMsgRecvSetWatch saves the watch status and sends a MON_CLIENT_MSG_RECV_SET_WATCH message to the <i>client_name</i> RTclient. The watch status controls whether the RTclient receives a MON_CLIENT_MSG_RECV_STATUS message each time a received message is inserted or deleted from the message queue for the connection to RTserver in the watched RTclient.</p> |

The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. If *msg\_type\_name* is `T_IPC_MON_ALL`, then all message types in the watched RTclient are watched.

Each `MON_CLIENT_MSG_RECV_STATUS` message contains six fields:

- a STR field containing the name of the RTclient
- a MSG field containing the received message
- an INT4 field containing whether the message was just inserted into or deleted from the connection's message queue (really of type `T_BOOL`); a value of `TRUE` means the message was inserted, a value of `FALSE` means the message was deleted
- an INT4 field containing the position in the queue where the received message was just inserted or deleted (a value of 0 means the front of the queue)
- a BINARY field containing a unique 8-byte identifier for the received message; this field is useful for processes that need to accurately track the entire contents of the connection's message queue, such as the RTmon GDI
- a BINARY field containing an array of unique 8-byte identifiers for all messages in the connection's message queue; this field is useful for processes that need to accurately track the entire contents of the connection's message queue, such as the RTmon GDI

In addition to sending a `MON_CLIENT_MSG_RECV_STATUS` message each time there is a message queue change, the watched RTclient also sends an initial `MON_CLIENT_MSG_RECV_STATUS` message when the watch status is set to `TRUE`, so that the RTclient does not have to wait for the next message queue change to receive the first `MON_CLIENT_MSG_RECV_STATUS` message.

To see if this RTclient is watching RTclient received messages, use `TpcSrvMonClientMsgRecvGetWatch`.

For an example of how `MON_CLIENT_MSG_RECV_STATUS` messages can be used, see the RTmon GDI Watch Messages Received window.

**Caution** `TpcSrvMonClientMsgRecvSetWatch` sends a `MON_CLIENT_MSG_RECV_SET_WATCH` message to RTserver, but does not explicitly flush the message. See `TpcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no initial status message if the RTclient named *client\_name* or the message type named *msg\_type\_name* does not exist.

RTclient received messages can only be watched, not polled.

The `MON_CLIENT_BUFFER_STATUS`, `MON_CLIENT_MSG_RECV_STATUS`, and `MON_CLIENT_MSG_SEND_STATUS` message types cannot be watched and are silently ignored by the internal monitoring callbacks that generate outgoing `MON_CLIENT_MSG_RECV_STATUS` messages. The callbacks ignore these message types to prevent infinite loops.

**See Also** `TipcSrvMonClientMsgRecvGetWatch`

**Examples** This uses `TipcSrvMonClientMsgRecvSetWatch` to start watching all received messages in the primary RTclient and creates a callback to process the incoming `MON_CLIENT_MSG_RECV_STATUS` messages:

```
/* ===== */
/*..process_mon_client_msg_recv_status -- process a MON_CLIENT_MSG_RECV_STATUS
message */
void T_ENTRY process_mon_client_msg_recv_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_IPC_MSG recv_msg;
 T_INT4 insert_flag;
 T_INT4 queue_pos;
 T_PTR msg_id;
 T_INT4 msg_id_size;
 T_PTR queue_msg_id_array;
 T_INT4 queue_msg_id_array_size;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_MSG, &recv_msg,
 T_IPC_FT_INT4, &insert_flag,
 T_IPC_FT_INT4, &queue_pos,
 T_IPC_FT_BINARY, &msg_id, &msg_id_size,
 T_IPC_FT_BINARY, &queue_msg_id_array,
 &queue_msg_id_array_size,
 NULL)) {
 return; /* error */
 }
}
```

```

 TutOut("Got CLIENT_MSG_RECV status.\n");
 TutOut("RTclient name = %s\n", client_name);
 TutOut("recv msg\n");
 if (!TipcMsgPrint(recv_msg, TutOut)) {
 return; /* error */
 }
 TutOut("insert flag = %s\n", insert_flag ? "TRUE" : "FALSE");
 TutOut("queue pos = %d\n", queue_pos);
 /* At this point, msg_id and queue_msg_id_array could be used */
 /* to accurately track any changes to the message queue (as */
 /* RTmon GDI does). */
} /* process_mon_client_msg_recv_status */
/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonClientMsgRecvSetWatch(srv, "primary",
T_IPC_MON_ALL,
 TRUE)) {

 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_MSG_RECV_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_msg_recv_status,
 NULL) == NULL) {

 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages.*/

```



# TipcSrvMonClientMsgSendGetWatch

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientMsgSendGetWatch — determine if this RTclient is watching sent messages in an RTclient                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientMsgSendGetWatch(srv, client_name, msg_type_name, watch_status_return) T_IPC_SRV srv; T_STR client_name; T_STR msg_type_name; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope)</p> <p><i>msg_type_name</i> — name of message type to get watch status for (use T_IPC_MON_ALL to indicate a watch of all message types)</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                               |
| <b>Return Values</b> | TRUE if the RTclient sent messages watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientMsgSendGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>T_ERR_NULL_PTR — <i>srv</i>, <i>client_name</i>, <i>msg_type_name</i>, or <i>watch_status_return</i> was null</li> </ul>                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b>   | <p>TipcSrvMonClientMsgSendGetWatch determines whether this RTclient is watching sent messages in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_MSG_SEND_STATUS message each time a message is sent to RTserver in the watched RTclient.</p> <p>If <i>msg_type_name</i> is T_IPC_MON_ALL, then all message types in the watched RTclient are watched.</p> |

Each MON\_CLIENT\_MSG\_SEND\_STATUS message contains two fields:

- a STR field containing the name of the RTclient
- a MSG field containing the sent message

To set whether this RTclient is watching RTclient sent messages, use TipcSrvMonClientMsgSendSetWatch.

For an example of how MON\_CLIENT\_MSG\_SEND\_STATUS messages can be used, see the RTmon GDI Watch Messages Sent window.

**Caution** If TipcSrvMonClientMsgSendGetWatch returns FALSE, it does not store a value in *watch\_status\_return*.

RTclient sent messages can only be watched, not polled.

**See Also** TipcSrvMonClientMsgSendSetWatch

**Examples** This example uses TipcSrvMonClientMsgSendGetWatch to get the status of watching all sent messages in the primary RTclient:

```
T_BOOL watch_status;

if (!TipcSrvMonClientMsgSendGetWatch(srv, "primary",
T_IPC_MON_ALL,
 &watch_status)) {
 return; /* error */
}
TutOut("This process is %s all messages sent in primary.\n",
 watch_status ? "watching" : "not watching");
```

## TipcSrvMonClientMsgSendSetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientMsgSendSetWatch — start or stop watching RTclient sent messages                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientMsgSendSetWatch(srv, client_name, msg_type_name, watch_status) T_IPC_SRV srv; T_STR client_name; T_STR msg_type_name; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to set watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope)</p> <p><i>msg_type_name</i> — name of message type to set watch status for (use T_IPC_MON_ALL to indicate a watch of all message types)</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Return Values</b> | TRUE if the RTclient sent messages watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientMsgSendSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i>, <i>client_name</i>, or <i>msg_type_name</i> was null</li> <li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li> <li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>   | <p>TipcSrvMonClientMsgSendSetWatch sets whether this RTclient is watching sent messages in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. TipcSrvMonClientMsgSendSetWatch saves the watch status and sends a MON_CLIENT_MSG_SEND_SET_WATCH message to the <i>client_name</i> RTclient. The watch status controls whether the RTclient receives a MON_CLIENT_MSG_SEND_STATUS message each time a message is sent to RTserver in the watched RTclient. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. If <i>msg_type_name</i> is T_IPC_MON_ALL then all message types in the watched RTclient are watched.</p> |

Each `MON_CLIENT_MSG_SEND_STATUS` message contains two fields:

- a `STR` field containing the name of the `RTclient`
- a `MSG` field containing the sent message

Unlike most of the other watch categories, the watched `RTclient` does not also send an initial `MON_CLIENT_MSG_SEND_STATUS` message when the watch status is set to `TRUE`. This is because the status message is only sent when a message is sent, and thus there is no current status to use for initialization.

To determine if this `RTclient` is watching `RTclient` sent messages, use `TipcSrvMonClientMsgSendGetWatch`.

For an example of how `MON_CLIENT_MSG_SEND_STATUS` messages can be used, see the `RTmon GDI Watch Messages Sent` window.

**Caution** `TipcSrvMonClientMsgSendSetWatch` sends a `MON_CLIENT_MSG_SEND_SET_WATCH` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

`RTclient` sent messages can only be watched, not polled.

The `MON_CLIENT_BUFFER_STATUS`, `MON_CLIENT_MSG_RECV_STATUS`, and `MON_CLIENT_MSG_SEND_STATUS` message types cannot be watched and are silently ignored by the internal monitoring callbacks that generate outgoing `MON_CLIENT_MSG_SEND_STATUS` messages. The callbacks ignore these message types to prevent infinite loops.

**See Also** `TipcSrvMonClientMsgSendGetWatch`

**Examples** This example uses `TipcSrvMonClientMsgSendSetWatch` to start watching all sent messages in the primary `RTclient` and creates a callback to process the incoming `MON_CLIENT_MSG_SEND_STATUS` messages:

```
/* ===== */
/*..process_mon_client_msg_send_status -- process a MON_CLIENT_MSG_SEND_STATUS
message */

void T_ENTRY process_mon_client_msg_send_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_IPC_MSG send_msg;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }
}
```

```

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_MSG, &send_msg,
 NULL)) {
 return; /* error */
 }

 TutOut("Got CLIENT_MSG_SEND status.\n");
 TutOut("RTclient name = %s\n", client_name);
 TutOut("send msg\n");
 if (!TipcMsgPrint(send_msg, TutOut)) {
 return; /* error */
 }
} /* process_mon_client_msg_send_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonClientMsgSendSetWatch(srv, "primary_rtie",
T_IPC_MON_ALL,
 TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_MSG_SEND_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_msg_send_status,
 NULL) == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```

# TipcSrvMonClientMsgTrafficPoll

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonClientMsgTrafficPoll — poll once for RTclient message traffic information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Synopsis      | <pre>T_BOOL TipcSrvMonClientMsgTrafficPoll(<i>srv</i>, <i>client_name</i>) T_IPC_SRV <i>srv</i>; T_STR <i>client_name</i>;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll for message traffic information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Return Values | TRUE if the poll for RTclient message traffic information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Diagnostics   | <p>If TipcSrvMonClientMsgTrafficPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>T_ERR_NULL_PTR — <i>srv</i> or <i>client_name</i> was null</li><li>any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Description   | <p>TipcSrvMonClientMsgTrafficPoll polls for RTclient message traffic information by sending a MON_CLIENT_MSG_TRAFFIC_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_MSG_TRAFFIC_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_CLIENT_MSG_TRAFFIC_POLL_RESULT message contains five fields:</p> <ul style="list-style-type: none"><li>a STR field containing the name of the RTclient</li><li>an INT4 field containing the number of messages the polled RTclient connection has sent</li><li>an INT4 field containing the number of messages the polled RTclient connection has received</li></ul> |

- an INT4 field containing the number of bytes the polled RTclient connection has sent
- an INT4 field containing the number of bytes the polled RTclient connection has received



In version 6.7 and later, the poll also returns these 8-byte variants of the previous fields. When these fields are present, they can be more accurate than their 4-byte counterparts (which may truncate the data):

- an INT8 field containing the number of messages the polled RTclient connection has sent
- an INT8 field containing the number of messages the polled RTclient connection has received
- an INT8 field containing the number of bytes the polled RTclient connection has sent
- an INT8 field containing the number of bytes the polled RTclient connection has received

There are multiple responses only if *client\_name* is a wildcard or T\_IPC\_MON\_ALL.

To poll once for message traffic information by message type, use TipcSrvMonClientMsgTypePoll. To poll once for message traffic information by subject, use TipcSrvMonClientSubjectPoll.

To poll once for RTserver message traffic information, use TipcSrvMonServerMsgTrafficPoll.

For an example of how MON\_CLIENT\_MSG\_TRAFFIC\_POLL\_RESULT messages can be used, see the RTmon GDI Client Message Traffic window.

**Caution** TipcSrvMonClientMsgTrafficPoll sends a MON\_CLIENT\_MSG\_TRAFFIC\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client\_name* does not exist; the poll is silently dropped.

RTclient message traffic information can only be polled, not watched.

**See Also** TipcSrvMonClientSubjectPoll, TipcSrvMonClientMsgTypePoll, TipcSrvMonServerMsgTrafficPoll, TipcSrvConnMsgWrite

**Examples** This example uses TpcSrvMonClientMsgTrafficPoll to poll for message traffic information in all RTclients (including this one), creates a callback to process the incoming MON\_CLIENT\_MSG\_TRAFFIC\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_msg_traffic_poll_result -- process a
MON_CLIENT_MSG_TRAFFIC_POLL_RESULT message */

void T_ENTRY process_mon_client_msg_traffic_poll_result
(
 T_IPC_CONN conn,
 T_IPC_CONN_PROCESS_CB_DATA data,
 T_CB_ARG arg
)
{
 T_STR client_name;
 T_INT4 total_msg_send_4;
 T_INT4 total_msg_recv_4;
 T_INT4 total_byte_send_4;
 T_INT4 total_byte_recv_4;
 T_INT8 total_msg_send_8;
 T_INT8 total_msg_recv_8;
 T_INT8 total_byte_send_8;
 T_INT8 total_byte_recv_8;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg, T_IPC_FT_STR, &client_name,
 T_IPC_FT_INT4, &total_msg_send_4,
 T_IPC_FT_INT4, &total_msg_recv_4,
 T_IPC_FT_INT4, &total_byte_send_4,
 T_IPC_FT_INT4, &total_byte_recv_4,
 NULL)) {
 return; return; /* error */
 }

 if (TipcMsgRead(data->msg, T_IPC_FT_INT8, &total_msg_recv_8,
 T_IPC_FT_INT8, &total_msg_send_8,
 T_IPC_FT_INT8, &total_byte_recv_8,
 T_IPC_FT_INT8, &total_byte_send_8,
 NULL)) {
 TutOut("Got CLIENT_MSG_TRAFFIC poll result (64-bit data).\n");
 TutOut("RTclient name = %s\n", client_name);
 }
}
```



```

/*
 * IMPORTANT: The format string for T_INT8 differs by platform.
 * The T_INT8_SPEC utility macro supplies the correct format
 * string for each platform.
 */
 TutOut("Total msg recv = " T_INT8_SPEC "\n",
total_msg_recv_8);
 TutOut("Total msg sent = " T_INT8_SPEC "\n",
total_msg_send_8);
 TutOut("Total byte recv = " T_INT8_SPEC "\n",
total_byte_recv_8);
 TutOut("Total byte sent = " T_INT8_SPEC "\n",
total_byte_recv_8);
}
else {
 TutOut("Got CLIENT_MSG_TRAFFIC poll result (32-bit data).\n");
 TutOut("RTclient name = %s\n", client_name);
 TutOut("Total msg recv = " T_INT4_SPEC "\n",
total_msg_recv_4);
 TutOut("Total msg sent = " T_INT4_SPEC "\n",
total_msg_send_4);
 TutOut("Total byte recv = " T_INT4_SPEC "\n",
total_byte_recv_4);
 TutOut("Total byte sent = " T_INT4_SPEC "\n",
total_byte_recv_4);
}
} /* process_mon_client_msg_traffic_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_MSG_TRAFFIC_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}
if (TipcSrvConnProcessCbCreate(srv, mt,
process_mon_client_msg_traffic_poll_result,
NULL) == NULL) {
 return; /* error */
}

/* send the poll request out to RTserver */
if (!TipcSrvMonClientMsgTrafficPoll(srv, "/...")) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

# TipcSrvMonClientMsgTypeExPoll

|                                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name                                                                              | TipcSrvMonClientMsgTypeExPoll — poll once for RTclient message type information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|  | This new function supersedes <code>TipcSrvMonClientMsgTypePoll</code> for clients that use release 6.7 or later.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Synopsis                                                                          | <pre>T_BOOL TipcSrvMonClientMsgTypeExPoll(<i>srv</i>, <i>client_name</i>, <i>msg_type_name</i>) T_IPC_SRV <i>srv</i>; T_STR <i>client_name</i>; T_STR <i>msg_type_name</i>;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Arguments                                                                         | <p><i>client_name</i> — name of RTclient to poll for message type information (wildcard names allowed, or use <code>T_IPC_MON_ALL</code> to indicate a poll of all RTclients matching the value of <code>Monitor_Scope</code>)</p> <p><i>srv</i> — connection handle to RTserver</p> <p><i>msg_type_name</i> — name of message type to poll for information (use <code>T_IPC_MON_ALL</code> to indicate a poll of all message types)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Return Values                                                                     | TRUE if the poll for RTclient message type information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Diagnostics                                                                       | <p>If <code>TipcSrvMonClientMsgTypeExPoll</code> fails, it returns FALSE and sets the global <code>SmartSockets</code> error number to one of:</p> <ul style="list-style-type: none"><li>• <code>T_ERR_NULL_PTR</code> — <i>srv</i>, <i>client_name</i> or <i>msg_type_name</i> was null</li><li>• any error number from <code>TipcSrvConnMsgWrite</code></li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Description                                                                       | <p><code>TipcSrvMonClientMsgTypeExPoll</code> polls for RTclient message type information by sending a <code>MON_CLIENT_MSG_TYPE_EX_POLL_CALL</code> message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or <code>T_IPC_MON_ALL</code> to poll all RTclients in the project that match the value of the <code>Monitor_Scope</code> option. The polled RTclient responds by sending back a <code>MON_CLIENT_MSG_TYPE_EX_POLL_RESULT</code> message. The response may or may not come back quickly, depending on what the polled RTclient is doing. If <i>msg_type_name</i> is <code>T_IPC_MON_ALL</code>, then all message types in the polled RTclient are polled. The polling RTclient can use functions such as <code>TipcSrvMsgSearchType</code> or <code>TipcSrvMainLoop</code> to get the response.</p> |

Each `MON_CLIENT_MSG_TYPE_EX_POLL_RESULT` message contains one fixed field and fifteen array fields. The fixed field is a `STR` field containing the name of the `RTclient`. The array fields are:

- a `STR_ARRAY` field containing the names of the message types
- an `INT4_ARRAY` field containing the message type numbers
- a `STR_ARRAY` field containing the message type grammars
- an `INT4_ARRAY` field containing whether the message type priorities are known (each array item is really of type `T_BOOL`)
- an `INT2_ARRAY` field containing the message type priorities (not applicable if the corresponding item in the previous array is `FALSE`)
- an `INT4_ARRAY` field containing the message type delivery modes (each array item is really of type `T_IPC_DELIVERY_MODE`)
- an `INT4_ARRAY` field containing the message type user-defined properties
- an `INT4_ARRAY` field containing the number of read callbacks the polled `RTclient` connection has for each message type
- an `INT4_ARRAY` field containing the number of write callbacks the polled `RTclient` connection has for each message type
- an `INT4_ARRAY` field containing the number of process callbacks the polled `RTclient` connection has for each message type
- an `INT4_ARRAY` field containing the number of queue callbacks the polled `RTclient` connection has for each message type
- an `INT8_ARRAY` field containing the number of messages the polled `RTclient` connection has sent for each message type
- an `INT8_ARRAY` field containing the number of messages the polled `RTclient` connection has received for each message type
- an `INT8_ARRAY` field containing the number of bytes the polled `RTclient` connection has sent for each message type
- an `INT8_ARRAY` field containing the number of bytes the polled `RTclient` connection has received for each message type

There are multiple responses only if *client\_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for global message traffic information, use `TipcSrvMonClientMsgTrafficExPoll`. To poll once for message traffic information by subject, use `TipcSrvMonClientSubjectExPoll`. To poll once for global callback information, use `TipcSrvMonClientCbPoll`.

**Caution** TipcSrvMonClientMsgTypeExPoll sends a MON\_CLIENT\_MSG\_TYPE\_EX\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client\_name* or the message type named *msg\_type\_name* does not exist; the poll is silently dropped.

RTclient message type information can only be polled, not watched.

There is no way to poll for RTserver message type information.

**See Also** TipcSrvMonClientCbPoll, TipcSrvMonClientSubjectExPoll, TipcSrvMonClientMsgTrafficPoll, TipcSrvConnMsgWrite

**Examples** This example uses TipcSrvMonClientMsgTypeExPoll to poll for all message type information in all RTclients (including this one), creates a callback to process the incoming MON\_CLIENT\_MSG\_TYPE\_EX\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results.

```
/* ===== */
/*..process_mon_client_msg_type_ex_poll_result -- process a
MON_CLIENT_MSG_TYPE_EX_POLL_RESULT message */

void process_mon_client_msg_type_ex_poll_result
(
 T_IPC_CONN conn,
 T_IPC_CONN_PROCESS_CB_DATA data,
 T_CB_ARG arg
)
{
 T_MSG msg;
 T_STR client_name;
 T_INT4 counter;
 T_STR delivery_mode_str = "";

 /* So many arrays... */
 T_STR *msg_type_name;
 T_INT4 *msg_num;
 T_STR *grammar;
 T_INT4 *priority_known; /* really (T_BOOL) */
 T_INT2 *priority; /* really (T_BOOL) */
 T_INT4 *delivery_mode; /* really T_IPC_DELIVERY_MODE */
 T_INT4 *user_prop;
 T_INT4 *num_read_cb;
 T_INT4 *num_write_cb;
 T_INT4 *num_process_cb;
 T_INT4 *num_queue_cb;
 T_INT8 *total_msg_send;
 T_INT8 *total_msg_rcv;
 T_INT8 *total_byte_send;
 T_INT8 *total_byte_rcv;
```

```

T_INT4 num_msg_type_name;
T_INT4 num_msg_num;
T_INT4 num_grammar;
T_INT4 num_priority_known;
T_INT4 num_priority;
T_INT4 num_delivery_mode;
T_INT4 num_user_prop;
T_INT4 num_num_read_cb;
T_INT4 num_num_write_cb;
T_INT4 num_num_process_cb;
T_INT4 num_num_queue_cb;
T_INT4 num_msg_send;
T_INT4 num_msg_recv;
T_INT4 num_byte_send;
T_INT4 num_byte_recv;

/* must set current field first */
if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
}

/* get the fields from the message */
msg = data->msg;
if (!TipcMsgNextStr(msg, &client_name)
 || !TipcMsgNextStrArray(msg, &msg_type_name, &num_msg_type_name)
 || !TipcMsgNextInt4Array(msg, &msg_num, &num_msg_num)
 || !TipcMsgNextStrArray(msg, &grammar, &num_grammar)
 || !TipcMsgNextInt4Array(msg, &priority_known,
&num_priority_known)
 || !TipcMsgNextInt2Array(msg, &priority, &num_priority)
 || !TipcMsgNextInt4Array(msg, &delivery_mode,
&num_delivery_mode)
 || !TipcMsgNextInt4Array(msg, &user_prop, &num_user_prop)
 || !TipcMsgNextInt4Array(msg, &num_read_cb, &num_num_read_cb)
 || !TipcMsgNextInt4Array(msg, &num_write_cb, &num_num_write_cb)
 || !TipcMsgNextInt4Array(msg, &num_process_cb,
&num_num_process_cb)
 || !TipcMsgNextInt4Array(msg, &num_queue_cb, &num_num_queue_cb)
 || !TipcMsgNextInt8Array(msg, &total_msg_send, &num_msg_send)
 || !TipcMsgNextInt8Array(msg, &total_msg_recv, &num_msg_recv)
 || !TipcMsgNextInt8Array(msg, &total_byte_send, &num_byte_send)
 || !TipcMsgNextInt8Array(msg, &total_byte_recv, &num_byte_recv))
{
 return; return; /* error */
}

TutOut("Got CLIENT_MSG_TYPE_EX poll result.\n");
TutOut("RTclient name = %s\n", client_name);

```

```

/* loop through all groups */
for (counter=0; counter<num_msg_type_name; counter++) {
 TutOut("message type name = %s\n", msg_type_name[counter]);
 TutOut("num = %d\n", msg_num[counter]);
 TutOut("grammar = %s\n", grammar[counter]);
 if (priority_known[counter]) {
 TutOut("priority = %d\n", priority[counter]);
 }
 else {
 TutOut("priority = unknown\n");
 }
 TipcDeliveryModeToStr(delivery_mode[counter],
&delivery_mode_str);
 TutOut("delivery mode = %s\n", delivery_mode_str);
 TutOut("user prop = " T_INT4_SPEC "\n",
 user_prop[counter]);
 TutOut("num read cb = " T_INT4_SPEC "\n",
 num_read_cb[counter]);
 TutOut("num write cb = " T_INT4_SPEC "\n",
 num_write_cb[counter]);
 TutOut("num process cb = " T_INT4_SPEC "\n",
 num_process_cb[counter]);
 TutOut("num queue cb = " T_INT4_SPEC "\n",
 num_queue_cb[counter]);
/*
* IMPORTANT: The format string for T_INT8 differs by platform.
* The T_INT8_SPEC utility macro supplies the correct format
* string for each platform.
*/
 TutOut("total msg send = " T_INT8_SPEC "\n",
 total_msg_send[counter]);
 TutOut("total msg recv = " T_INT8_SPEC "\n",
 total_msg_recv[counter]);
 TutOut("total byte send = " T_INT8_SPEC "\n",
 total_byte_send[counter]);
 TutOut("total byte recv = " T_INT8_SPEC "\n",
 total_byte_recv[counter]);
}
} /* process_mon_client_msg_type_poll_result */

```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_MSG_TYPE_EX_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}
if (TipcSrvConnProcessCbCreate(srv, mt,

process_mon_client_msg_type_ex_poll_result,
 NULL) == NULL) {

 return; /* error */
}

/* send the poll request out to RTserver */
if (!TipcSrvMonClientMsgTypeExPoll(srv, "/...", T_IPC_MON_ALL)) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

# TipcSrvMonClientMsgTypePoll

---

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonClientMsgTypePoll — poll once for RTclient message type information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Synopsis      | <pre>T_BOOL TipcSrvMonClientMsgTypePoll(srv, client_name, msg_type_name) T_IPC_SRV srv; T_STR client_name; T_STR msg_type_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll for message type information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p> <p><i>msg_type_name</i> — name of message type to poll for information (use T_IPC_MON_ALL to indicate a poll of all message types)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Return Values | TRUE if the poll for RTclient message type information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Diagnostics   | <p>If TipcSrvMonClientMsgTypePoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i>, <i>client_name</i> or <i>msg_type_name</i> was null</li><li>• any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Description   | <p>TipcSrvMonClientMsgTypePoll polls for RTclient message type information by sending a MON_CLIENT_MSG_TYPE_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_MSG_TYPE_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. If <i>msg_type_name</i> is T_IPC_MON_ALL, then all message types in the polled RTclient are polled. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> |



Each `MON_CLIENT_MSG_TYPE_POLL_RESULT` message contains one fixed field and one or more groups of fifteen fields. The fixed field is a STR field containing the name of the RTclient. The group of fifteen fields is:

- a STR field containing the name of the message type
- an INT4 field containing the message type number
- a STR field containing the message type grammar
- an INT4 field containing whether the message type priority is known (really of type `T_BOOL`)
- an INT2 field containing the message type priority (not applicable if the previous field is `FALSE`)
- an INT4 field containing the message type delivery mode (really of type `T_IPC_DELIVERY_MODE`)
- an INT4 field containing the message type user-defined property
- an INT4 field containing the number of read callbacks the polled RTclient connection has for this message type
- an INT4 field containing the number of write callbacks the polled RTclient connection has for this message type
- an INT4 field containing the number of process callbacks the polled RTclient connection has for this message type
- an INT4 field containing the number of queue callbacks the polled RTclient connection has for this message type
- an INT4 field containing the number of messages the polled RTclient connection has sent for this message type
- an INT4 field containing the number of messages the polled RTclient connection has received for this message type
- an INT4 field containing the number of bytes the polled RTclient connection has sent for this message type
- an INT4 field containing the number of bytes the polled RTclient connection has received for this message type

There are multiple responses only if *client\_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for global message traffic information, use `TipcSrvMonClientMsgTrafficPoll`. To poll once for message traffic information by subject, use `TipcSrvMonClientSubjectPoll`. To poll once for global callback information, use `TipcSrvMonClientCbPoll`.

**Caution** TipcSrvMonClientMsgTypePoll sends a `MON_CLIENT_MSG_TYPE_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTclient named *client\_name* or the message type named *msg\_type\_name* does not exist; the poll is silently dropped.

RTclient message type information can only be polled, not watched.

There is no way to poll for RTserver message type information.

**See Also** `TipcSrvMonClientCbPoll`, `TipcSrvMonClientSubjectPoll`, `TipcSrvMonClientMsgTrafficPoll`, `TipcSrvConnMsgWrite`

**Examples** This example uses `TipcSrvMonClientMsgTypePoll` to poll for all message type information in all RTclients (including this one), creates a callback to process the incoming `MON_CLIENT_MSG_TYPE_POLL_RESULT` messages, and then waits up to 10 seconds for the poll results.

```
/* ===== */
/*..process_mon_client_msg_type_poll_result -- process a
MON_CLIENT_MSG_TYPE_POLL_RESULT message */
void process_mon_client_msg_type_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_STR msg_type_name;
 T_INT4 num;
 T_STR grammar;
 T_INT4 priority_known; /* really (T_BOOL) */
 T_INT2 priority; /* not applicable if priority_known is FALSE */
 T_INT4 delivery_mode; /* really T_IPC_DELIVERY_MODE */
 T_INT4 user_prop;
 T_INT4 num_read_cb;
 T_INT4 num_write_cb;
 T_INT4 num_process_cb;
 T_INT4 num_queue_cb;
 T_INT4 total_msg_send;
 T_INT4 total_msg_rcv;
 T_INT4 total_byte_send;
 T_INT4 total_byte_rcv;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }
}
```

```

/* get first fixed field */
if (!TipcMsgNextStr(data->msg, &client_name)) {
return; /* error */
}
TutOut("Got CLIENT_MSG_TYPE poll result.\n");
TutOut("RTclient name = %s\n", client_name);

/* loop through all groups */
while (TipcMsgRead(data->msg,
 T_IPC_FT_STR, &msg_type_name,
 T_IPC_FT_INT4, &num,
 T_IPC_FT_STR, &grammar,
 T_IPC_FT_INT4, &priority_known,
 T_IPC_FT_INT2, &priority,
 T_IPC_FT_INT4, &delivery_mode,
 T_IPC_FT_INT4, &user_prop,
 T_IPC_FT_INT4, &num_read_cb,
 T_IPC_FT_INT4, &num_write_cb,
 T_IPC_FT_INT4, &num_process_cb,
 T_IPC_FT_INT4, &num_queue_cb,
 T_IPC_FT_INT4, &total_msg_send,
 T_IPC_FT_INT4, &total_msg_recv,
 T_IPC_FT_INT4, &total_byte_send,
 T_IPC_FT_INT4, &total_byte_recv,
 NULL)) {
 TutOut("message type name = %s\n", msg_type_name);
 TutOut("num = %d\n", num);
 TutOut("grammar = %s\n", grammar);
 if (priority_known) {
 TutOut("priority = %d\n", priority);
 }
 else {
 TutOut("priority = unknown\n");
 }
 TutOut("delivery mode = %d\n", delivery_mode);
 TutOut("user prop = %d\n", user_prop);
 TutOut("num read cb = %d\n", num_read_cb);
 TutOut("num write cb = %d\n", num_write_cb);
 TutOut("num process cb = %d\n", num_process_cb);
 TutOut("num queue cb = %d\n", num_queue_cb);
 TutOut("total msg send = %d\n", total_msg_send);
 TutOut("total msg recv = %d\n", total_msg_recv);
 TutOut("total byte send = %d\n", total_byte_send);
 TutOut("total byte recv = %d\n", total_byte_recv);
}

/* make sure we reached the end of the message */
if (TutErrNumGet() != T_ERR_MSG_EOM) {
 return; /* error */
}
} /* process_mon_client_msg_type_poll_result */

```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientMsgTypePoll(srv, "/...", T_IPC_MON_ALL)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_MSG_TYPE_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
process_mon_client_msg_type_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonClientNamesGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientNamesGetWatch — determine if this RTclient is watching RTclient names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientNamesGetWatch(srv, watch_status_return) T_IPC_SRV srv; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Return Values</b> | TRUE if the RTclient names watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientNamesGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>T_ERR_NULL_PTR — <i>srv</i> or <i>watch_status_return</i> was null</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Description</b>   | <p>TipcSrvMonClientNamesGetWatch determines whether this RTclient is watching the RTclient names in the current project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_NAMES_STATUS message each time an RTclient is created or destroyed. An RTclient is considered created when it connects to RTserver. An RTclient is considered destroyed when it disconnects from RTserver.</p> <p>Each MON_CLIENT_NAMES_STATUS message contains five fields:</p> <ul style="list-style-type: none"> <li>a STR_ARRAY field containing the current RTclient names</li> <li>a STR_ARRAY field containing RTclient info strings of the form "<i>ident: user@node</i>" (such as "Runtime_RTie: ssuser@workstation1")</li> <li>a STR_ARRAY field containing the name of the RTserver each RTclient is connected to</li> <li>a STR field containing the name of the created RTclient (or "" if not applicable)</li> <li>a STR field containing the name of the destroyed RTclient (or "" if not applicable)</li> </ul> <p>To set whether this RTclient is watching RTclient names, use TipcSrvMonClientNamesSetWatch. To poll once for the RTclient names, use TipcSrvMonClientNamesPoll.</p> |
| <b>Caution</b>       | If TipcSrvMonClientNamesGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

**See Also**     TipcSrvMonClientNamesPoll, TipcSrvMonClientNamesSetWatch

**Examples**     This example uses TipcSrvMonClientNamesGetWatch to get the status of watching RTclient names:

```
T_BOOL watch_status;

if (!TipcSrvMonClientNamesGetWatch(srv, &watch_status)) {
 return; /* error */
}
TutOut("This process is %s RTclient names.\n",
 watch_status ? "watching" : "not watching");
```

## TipcSrvMonClientNamesNumPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientNamesNumPoll — poll for the number of RTclients in the current project that match the value of the Monitor_Scope option                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientNamesNumPoll(srv) T_IPC_SRV srv;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>     | <i>srv</i> — connection handle to RTserver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Return Values</b> | TRUE if the poll for the number of RTclients was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientNamesNumPoll fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>any error number from TipcSrvMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b>   | <p>TipcSrvMonClientNamesNumPoll polls for the number of RTclients in the current project that match the value of the Monitor_Scope option by sending a MON_CLIENT_NAMES_NUM_POLL_CALL to the RTserver. The <i>srv</i> argument indicates the connection on which the poll is to be sent.</p> <p>The RTserver responds by sending back a MON_CLIENT_NAMES_NUM_POLL_RESULT message. Each MON_CLIENT_NAMES_NUM_POLL_RESULT message contains one field:</p> <ul style="list-style-type: none"> <li>an INT4 field containing the number of RTclients in the project that match Monitor_Scope</li> </ul> <p>If there are no RTclients in the project that match the value of Monitor_Scope, the MON_CLIENT_NAMES_NUM_POLL_RESULT message gives 0 for the number of clients.</p> |
| <b>Caution</b>       | <p>TipcSrvMonClientNamesNumPoll sends a MON_CLIENT_NAMES_NUM_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.</p> <p>The number of RTclients can only be polled, not watched.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>See Also</b>      | TipcSrvMonClientNamesPoll                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

**Examples** This example uses TipcSrvMonClientNamesNumPoll to poll for the number of RTclient names, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_INT4 num_client_names;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientNamesNumPoll(srv)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_NAMES_NUM_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvConnMsgSearchType(srv, 10.0, mt);
if (msg == NULL) {
 return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
 return; /* error */
}

if (!TipcMsgNextInt4(msg, &num_client_names)) {
 return; /* error */
}

TutOut("Got CLIENT_NAMES_NUM poll result.\n");
TutOut("number of client names = %d\n", num_client_names);

/* clean up */
if (!TipcMsgDestroy(msg)) {
 return; /* error */
}
```



## TipcSrvMonClientNamesPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientNamesPoll — poll once for RTclient names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientNamesPoll(srv) T_IPC_SRV srv;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>     | <i>srv</i> — connection handle to RTserver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Return Values</b> | TRUE if the poll for RTclient names was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientNamesPoll fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Description</b>   | <p>TipcSrvMonClientNamesPoll polls for the RTclient names in the current project that match the value of the Monitor_Scope option by sending a MON_CLIENT_NAMES_POLL_CALL message to RTserver. RTserver responds by sending back a MON_CLIENT_NAMES_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response. The MON_CLIENT_NAMES_POLL_RESULT message contains three fields:</p> <ul style="list-style-type: none"> <li>• a STR_ARRAY field containing the current RTclient names</li> <li>• a STR_ARRAY field containing RTclient info strings of the form "<i>ident: user@node</i>" (such as "Primary_RTclient: ssuser@workstation1")</li> <li>• a STR_ARRAY field containing the name of the RTserver each RTclient is connected to</li> </ul> <p>To watch for created and destroyed RTclient names, use TipcSrvMonClientNamesSetWatch.</p> |
| <b>Caution</b>       | TipcSrvMonClientNamesPoll sends a MON_CLIENT_NAMES_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>See Also</b>      | TipcSrvMonClientNamesGetWatch, TipcSrvMonClientNamesSetWatch, TipcSrvConnMsgWrite                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

**Examples** This example uses TpcSrvMonClientNamesPoll to poll for RTclient names, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *client_names;
T_INT4 num_client_names;
T_STR *info_strs;
T_STR *server_names;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TpcSrvMonClientNamesPoll(srv)) {
 return; /* error */
}

mt = TpcMtLookupByNum(T_MT_MON_CLIENT_NAMES_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TpcSrvConnMsgSearchType(srv, 10.0, mt);
if (msg == NULL) {
 return; /* error */
}

/* set the current field to the first field */
if (!TpcMsgSetCurrent(msg, 0)) {
 return; /* error */
}

/* client_names, info_strs, and server_names contains the same number of elements */
if (!TpcMsgNextStrArray(msg, &client_names, &num_client_names)
 || !TpcMsgNextStrArray(msg, &info_strs, &num_client_names)
 || !TpcMsgNextStrArray(msg, &server_names, &num_client_names))
{
 return; /* error */
}

TutOut("Got CLIENT_NAMES poll result.\n");
for (counter = 0; counter < num_client_names; counter++) {
 TutOut("client_names[%d] = %s\n", counter,
 client_names[counter]);
 TutOut("info_strs[%d] = %s\n", counter, info_strs[counter]);
 TutOut("server_names[%d] = %s\n", counter,
 server_names[counter]);
}

/* clean up */
if (!TpcMsgDestroy(msg)) {
 return; /* error */
}
```

# TipcSrvMonClientNamesSetWatch

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientNamesSetWatch — start or stop watching RTclient names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientNamesSetWatch(srv, watch_status) T_IPC_SRV srv; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Return Values</b> | TRUE if the RTclient names watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientNamesSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> was null</li><li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li><li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b>   | <p>TipcSrvMonClientNamesSetWatch sets whether this RTclient is watching the RTclient names in the current project that match the value of the Monitor_Scope option. TipcSrvMonClientNamesSetWatch saves the watch status and sends a MON_CLIENT_NAMES_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_CLIENT_NAMES_STATUS message each time an RTclient is created or destroyed. An RTclient is considered created when it connects to RTserver. An RTclient is considered destroyed when it disconnects from RTserver. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.</p> <p>The Srv_Client_Names_Min_Msgs option, described in the <i>TIBCO SmartSockets User's Guide</i>, controls the number of fields returned in the MON_CLIENT_NAMES_STATUS message. If Srv_Client_Names_Min_Msgs is set to FALSE, each MON_CLIENT_NAMES_STATUS message contains the following fields:</p> <ul style="list-style-type: none"><li>• a STR_ARRAY field containing the current RTclient names</li><li>• a STR_ARRAY field containing RTclient info strings of the form "<i>ident: user@node</i>" (such as "Primary_RTclient: ssuser@workstation1")</li><li>• a STR_ARRAY field containing the name of the RTserver each RTclient is connected to</li><li>• a STR field containing the name of the created RTclient (or "" if not applicable)</li></ul> |

- a STR field containing the name of the destroyed RTclient (or "" if not applicable)
- a STR field containing the reason for the client's disconnection (or "" if not applicable). This field is returned if the RTserver version is 6.8 or later. The possible disconnect reasons known at the time this document was written are:
  - **disconnect** - The client cleanly disconnected.
  - **failure detected** - A failure was detected while trying to communicate with the client (in most instances this means a keepalive failed due to network failure or a non-responsive client).
  - **max\_buffer exceeded** - The client was disconnected by the server because its write buffer exceeded the size allowed by the "client\_max\_buffer" option.
  - **socket closed** - The client unexpectedly closed the socket (an EOF was detected). This would typically occur when a client terminates without first disconnecting (e.g. was CTRL-C'd).
  - **nak failure** - The client was disconnected by the group manager due to unrecoverable loss / NAK failure (disconnect\_client\_on\_nak\_failure was set to TRUE).
  - **disconnected by group** - The client was disconnected by the group connection server (RTgms). This message is used only when the specific reason for the disconnect is not known or is unavailable (which should only happen with a 6.8+ RTserver and a pre-6.8 RTgms).
  - **group connection lost** - The connection to the client's group server (RTgms) was lost, either due to network failure or because the RTgms is no longer running, or because the RTserver disconnected the RTgms.
  - **server connection lost** - The connection to the client's server was lost, either due to network failure or because the server is no longer running.
  - **server out of memory** - The server dropped the connection because it ran out of memory while trying to read from or write to the connection.
  - **gms out of memory** - The gms dropped the connection because it ran out of memory while trying to read from or write to the connection.
  - **unknown** - The connection was closed for an unknown (or unhandled) reason.

In addition to sending a MON\_CLIENT\_NAMES\_STATUS message each time an RTclient is created or destroyed, RTserver also sends an initial MON\_CLIENT\_NAMES\_STATUS message when the watch status is set to TRUE, so that the RTclient does not have to wait for the next RTclient name change to

receive the first `MON_CLIENT_NAMES_STATUS` message. User's can limit the size of subsequent `MON_CLIENT_NAMES_STATUS` messages to only the name of the RTclient created or destroyed and the reason for the destroy by setting the RTserver option `Srv_Client_Names_Min_Msgs` to `TRUE`.

To determine if this RTclient is watching RTclient names, use `TipcSrvMonClientNamesGetWatch`. To poll once for the RTclient names, use `TipcSrvMonClientNamesPoll`.

**Caution** `TipcSrvMonClientNamesSetWatch` sends a `MON_CLIENT_NAMES_SET_WATCH` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

**See Also** `TipcSrvMonClientNamesGetWatch`, `TipcSrvMonClientNamesPoll`

**Examples** This example uses `TipcSrvMonClientNamesSetWatch` to start watching RTclient names and creates a callback to process the incoming `MON_CLIENT_NAMES_STATUS` messages:

```
/* ===== */
/*..process_mon_client_names_status -- process a MON_CLIENT_NAMES_STATUS message */
void T_ENTRY process_mon_client_names_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR *client_names;
 T_INT4 num_client_names;
 T_STR *info_strs;
 T_STR *server_names;
 T_STR created_client;
 T_STR destroyed_client;
 T_STR reason;
 T_INT4 counter;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
```

```

/* client_names, info_strs, and server_names contains the same
number of elements */

 if (!TipcMsgNextStrArray(data->msg, &client_names,
 &num_client_names)
 || !TipcMsgNextStrArray(data->msg, &info_strs,
 &num_client_names)
 || !TipcMsgNextStrArray(data->msg, &server_names,
 &num_client_names)
 || !TipcMsgNextStr(data->msg, &created_client)
 || !TipcMsgNextStr(data->msg, &destroyed_client)) {
 return; /* error */
 }

 TutOut("Got CLIENT_NAMES status.\n");
 for (counter = 0; counter < num_client_names; counter++) {
 TutOut("client_names[%d] = %s\n", counter,
 client_names[counter]);
 TutOut("info_strs[%d] = %s\n", counter, info_strs[counter]);
 TutOut("server_names[%d] = %s\n", counter,
 server_names[counter]);
 }
 TutOut("created RTclient = %s\n", created_client);
 TutOut("destroyed RTclient = %s\n", destroyed_client);
 if (TipcMsgNextStr(data->msg, &reason)) {
 TutOut("reason for disconnect = %s\n", reason);
 }
} /* process_mon_client_names_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonClientNamesSetWatch(srv, TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_NAMES_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_names_status, NULL)
 == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```

## TipcSrvMonClientOptionPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientOptionPoll — poll once for RTclient option information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientOptionPoll(srv, client_name, option_name) T_IPC_SRV srv; T_STR client_name; T_STR option_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll for option information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p> <p><i>option_name</i> — name of option to poll for information (use T_IPC_MON_ALL to indicate a poll of all options)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Return Values</b> | TRUE if the poll for RTclient option information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientOptionPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i>, <i>client_name</i>, or <i>option_name</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>   | <p>TipcSrvMonClientOptionPoll polls for RTclient option information by sending a MON_CLIENT_OPTION_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_OPTION_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. If <i>option_name</i> is T_IPC_MON_ALL, then all options in the polled RTclient are polled. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> |

Each `MON_CLIENT_OPTION_POLL_RESULT` message contains one fixed field followed by one or more groups of five fields. The fixed field is a `STR` field containing the name of the `RTclient`. The group of five fields is:

- a `STR` field containing the name of the option
- an `INT2` field containing the option type (really `T_OPT_TYPE`)
- a `STR` field containing the value of the option, converted to a string like the `setopt` command displays
- an `INT4` field containing whether the option is required to have a value (that is, it cannot be `UNKNOWN`)
- a `STR_ARRAY` field containing the legal values for the option; this array is empty if not applicable

There are multiple responses only if *client\_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for `RTserver` option information, use `TpcSrvMonServerOptionPoll`.

For an example of how `MON_CLIENT_OPTION_POLL_RESULT` messages can be used, see the `RTmon GDI Client Information` window.

**Caution** `TpcSrvMonClientOptionPoll` sends a `MON_CLIENT_OPTION_POLL_CALL` message to `RTserver`, but does not explicitly flush the message. See `TpcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no response message if the `RTclient` named *client\_name* or the option named *option\_name* does not exist; the poll is silently dropped.

`RTclient` option information can only be polled, not watched.

**See Also** `TpcSrvMonServerOptionPoll`, `TpcSrvConnMsgWrite`



**Examples** This example uses TipcSrvMonClientOptionPoll to poll for all option information in all RTclients (including this one), creates a callback to process the incoming MON\_CLIENT\_OPTION\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_option_poll_result -- process a
MON_CLIENT_OPTION_POLL_RESULT message */
void T_ENTRY process_mon_client_option_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_STR option_name;
 T_INT2 type; /* really (T_OPT_TYPE) */
 T_STR value_str;
 T_INT4 required; /* really (T_BOOL) */
 T_STR *legal_vals;
 T_INT4 num_legal_vals; /* is zero if legal_vals N/A */
 T_INT4 counter; /* loop counter */

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get first fixed field */
 if (!TipcMsgNextStr(data->msg, &client_name)) {
 return; /* error */
 }
 TutOut("Got CLIENT_OPTION poll result.\n");
 TutOut("RTclient name = %s\n", client_name);

 /* loop through all groups */
 while (TipcMsgRead(data->msg,
 T_IPC_FT_STR, &option_name,
 T_IPC_FT_INT2, &type,
 T_IPC_FT_STR, &value_str,
 T_IPC_FT_INT4, &required,
 T_IPC_FT_STR_ARRAY, &legal_vals,
&num_legal_vals,
 NULL)) {
 TutOut("option name = %s\n", option_name);
 TutOut("type = %d\n", type);
 TutOut("value str = %s\n", value_str);
 TutOut("required = %s\n", required ? "TRUE" : "FALSE");
 if (num_legal_vals > 0) {
 TutOut("legal vals:\n");
 for (counter = 0; counter < num_legal_vals; counter++) {
 TutOut(" %s\n", legal_vals[counter]);
 }
 }
 }
}
```

```

 /* make sure we reached the end of the message */
 if (TutErrNumGet() != T_ERR_MSG_EOM) {
 return; /* error */
 }
} /* process_mon_client_option_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientOptionPoll(srv, "...", T_IPC_MON_ALL)) {
 return; /* error */
}
mt = TipcMtLookupByNum(T_MT_MON_CLIENT_OPTION_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_option_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonClientSubjectExPoll

---

**Name** TipcSrvMonClientSubjectExPoll — poll once for RTclient subject information



This new function supersedes `TipcSrvMonClientSubjectPoll` for clients that use release 6.7 or later.

**Synopsis**

```
T_BOOL TipcSrvMonClientSubjectExPoll(srv, client_name, subject_name)
T_IPC_SRV srv;
T_STR client_name;
T_STR subject_name;
```

**Arguments** *srv* — connection handle to RTserver

*client\_name* — name of RTclient to poll for subject information (wildcard names allowed, or use `T_IPC_MON_ALL` to indicate a poll of all RTclients matching the value of `Monitor_Scope`)

*subject\_name* — name of subject to poll for information (use `T_IPC_MON_ALL` to indicate a poll of all subjects in the polled RTclient)

**Return Values** TRUE if the poll for RTclient subject information was successfully initiated, FALSE otherwise.

**Diagnostics** If `TipcSrvMonClientSubjectExPoll` fails, it returns FALSE and sets the global SmartSockets error number to one of:

- `T_ERR_NULL_PTR` — *srv*, *client\_name* or *subject\_name* was null
- any error number from `TipcSrvConnMsgWrite`

**Description** `TipcSrvMonClientSubjectExPoll` polls for RTclient subject information by sending a `MON_CLIENT_SUBJECT_EX_POLL_CALL` message to the *client\_name* RTclient. The *client\_name* argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or `T_IPC_MON_ALL` to poll all RTclients in the project that match the value of the `Monitor_Scope` option. The polled RTclient responds by sending back a `MON_CLIENT_SUBJECT_EX_POLL_RESULT` message. The response may or may not come back quickly, depending on what the polled RTclient is doing. If *subject\_name* is `T_IPC_MON_ALL`, then all subjects ever used in the polled RTclient are polled. The polling RTclient can use functions such as `TipcSrvMsgSearchType` or `TipcSrvMainLoop` to get the response.

Each `MON_CLIENT_SUBJECT_EX_POLL_RESULT` message contains one fixed field followed by five array fields. The fixed field is a `STR` field containing the name of the `RTclient`. The array fields are:

- a `STR_ARRAY` field containing the name of the subject
- an `INT8_ARRAY` field containing the number of messages the polled `RTclient` connection has sent to each subject
- an `INT8_ARRAY` field containing the number of messages the polled `RTclient` connection has received for each subject
- an `INT8_ARRAY` field containing the number of bytes the polled `RTclient` connection has sent for each subject
- an `INT8_ARRAY` field containing the number of bytes the polled `RTclient` connection has received for each subject

There are multiple responses only if *client\_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for global message traffic information, use `TipcSrvMonClientMsgTrafficPoll`. To poll once for message traffic information by message type, use `TipcSrvMonClientMsgTypeExPoll`.

For an example of how `MON_CLIENT_SUBJECT_EX_POLL_RESULT` messages can be used, see the `RTmon GDI Client Subject Message Traffic` window.

**Caution** `TipcSrvMonClientSubjectExPoll` sends a `MON_CLIENT_SUBJECT_EX_POLL_CALL` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvGetAutoFlushSize` for more information on message buffering.

There is no response message if the `RTclient` named *client\_name* or the subject named *subject\_name* does not exist; the poll is silently dropped.

`RTclient` subject information can only be polled, not watched.

There is no way to poll for `RTserver` subject information.

**See Also** `TipcSrvMonClientMsgTrafficPoll`, `TipcSrvConnMsgWrite`

**Examples** This example uses TipcSrvMonClientSubjectExPoll to poll for all subject information in all RTclients (including this one), creates a callback to process the incoming MON\_CLIENT\_SUBJECT\_POLL\_EX\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_subject_ex_poll_result -- process a
MON_CLIENT_SUBJECT_EX_POLL_RESULT message */

void process_mon_client_subject_ex_poll_result
(
 T_IPC_CONN conn,
 T_IPC_CONN_PROCESS_CB_DATA data,
 T_CB_ARG arg
)
{
 T_MSG msg;
 T_STR client_name;
 T_STR *subject_name;
 T_INT4 num_subjects;
 T_INT8 *total_msg_send;
 T_INT4 num_msg_send;
 T_INT8 *total_msg_rcv;
 T_INT4 num_msg_rcv;
 T_INT8 *total_byte_send;
 T_INT4 num_byte_send;
 T_INT8 *total_byte_rcv;
 T_INT4 num_byte_rcv;
 T_INT4 counter;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 msg = data->msg;
 if (!TipcMsgNextStr(msg, &client_name)
 || !TipcMsgNextStrArray(msg, &subject_name, &num_subjects)
 || !TipcMsgNextInt8Array(msg, &total_msg_send, &num_msg_send)
 || !TipcMsgNextInt8Array(msg, &total_msg_rcv, &num_msg_rcv)
 || !TipcMsgNextInt8Array(msg, &total_byte_send, &num_byte_send)
 || !TipcMsgNextInt8Array(msg, &total_byte_rcv, &num_byte_rcv))
 {
 return; return; /* error */
 }
 TutOut("Got CLIENT_SUBJECT_EX poll result.\n");
 TutOut("RTclient name = %s\n", client_name);

 /* loop through all groups */
 for (counter=0; counter<num_subjects; counter++) {
 TutOut("subject name = %s\n", subject_name[counter]);
 }
 /*
 * IMPORTANT: The format string for T_INT8 differs by platform.
 * The T_INT8_SPEC utility macro supplies the correct format
 */
}
```

```

 *string for each platform.
 */
 TutOut("total msg send = " T_INT8_SPEC "\n",
 total_msg_send[counter]);
 TutOut("total msg recv = " T_INT8_SPEC "\n",
 total_msg_recv[counter]);
 TutOut("total byte send = " T_INT8_SPEC "\n",
 total_byte_send[counter]);
 TutOut("total byte recv = " T_INT8_SPEC "\n",
 total_byte_recv[counter]);
}

} /* process_mon_client_subject_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBJECT_EX_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}
if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_subject_ex_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* send the poll request out to RTserver */
if (!TipcSrvMonClientSubjectExPoll(srv, "...", T_IPC_MON_ALL)) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonClientSubjectPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientSubjectPoll — poll once for RTclient subject information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientSubjectPoll(srv, client_name, subject_name) T_IPC_SRV srv; T_STR client_name; T_STR subject_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll for subject information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p> <p><i>subject_name</i> — name of subject to poll for information (use T_IPC_MON_ALL to indicate a poll of all subjects in the polled RTclient)</p>                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Return Values</b> | TRUE if the poll for RTclient subject information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientSubjectPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i>, <i>client_name</i>, or <i>subject_name</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b>   | <p>TipcSrvMonClientSubjectPoll polls for RTclient subject information by sending a MON_CLIENT_SUBJECT_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_SUBJECT_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. If <i>subject_name</i> is T_IPC_MON_ALL, then all subjects ever used in the polled RTclient are polled. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> |

Each `MON_CLIENT_SUBJECT_POLL_RESULT` message contains one fixed field followed by one or more groups of five fields. The fixed field is a STR field containing the name of the RTclient. The group of five fields is:

- a STR field containing the name of the subject
- an INT4 field containing the number of messages the polled RTclient connection has sent to this subject
- an INT4 field containing the number of messages the polled RTclient connection has received for this subject
- an INT4 field containing the number of bytes the polled RTclient connection has sent for this subject
- an INT4 field containing the number of bytes the polled RTclient connection has received for this subject

There are multiple responses only if *client\_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for global message traffic information, use `TpcSrvMonClientMsgTrafficPoll`. To poll once for message traffic information by message type, use `TpcSrvMonClientMsgTypePoll`.

For an example of how `MON_CLIENT_SUBJECT_POLL_RESULT` messages can be used, see the RTmon GDI Client Subject Message Traffic window.

**Caution** `TpcSrvMonClientSubjectPoll` sends a `MON_CLIENT_SUBJECT_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TpcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTclient named *client\_name* or the subject named *subject\_name* does not exist; the poll is silently dropped.

RTclient subject information can only be polled, not watched.

There is no way to poll for RTserver subject information.

**See Also** `TpcSrvMonClientMsgTrafficPoll`, `TpcSrvConnMsgWrite`



**Examples** This example uses TIPC\_SrvMonClientSubjectPoll to poll for all subject information in all RTclients (including this one), creates a callback to process the incoming MON\_CLIENT\_SUBJECT\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_subject_poll_result -- process a
MON_CLIENT_SUBJECT_POLL_RESULT message */
void T_ENTRY process_mon_client_subject_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_STR subject_name;
 T_INT4 total_msg_send;
 T_INT4 total_msg_rcv;
 T_INT4 total_byte_send;
 T_INT4 total_byte_rcv;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get first fixed field */
 if (!TipcMsgNextStr(data->msg, &client_name)) {
 return; /* error */
 }
 TutOut("Got CLIENT_SUBJECT poll result.\n");
 TutOut("RTclient name = %s\n", client_name);

 /* loop through all groups */
 while (TipcMsgRead(data->msg,
 T_IPC_FT_STR, &subject_name,
 T_IPC_FT_INT4, &total_msg_send,
 T_IPC_FT_INT4, &total_msg_rcv,
 T_IPC_FT_INT4, &total_byte_send,
 T_IPC_FT_INT4, &total_byte_rcv,
 NULL)) {
 TutOut("subject name = %s\n", subject_name);
 TutOut("total msg send = %d\n", total_msg_send);
 TutOut("total msg rcv = %d\n", total_msg_rcv);
 TutOut("total byte send = %d\n", total_byte_send);
 TutOut("total byte rcv = %d\n", total_byte_rcv);
 }

 /* make sure we reached the end of the message */
 if (TutErrNumGet() != T_ERR_MSG_EOM) {
 return; /* error */
 }
} /* process_mon_client_subject_poll_result */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientSubjectPoll(srv, "/...", T_IPC_MON_ALL)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBJECT_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}
if (TipcSrvConnProcessCbCreate(srv, mt,
process_mon_client_subject_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonClientSubscribeGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientSubscribeGetWatch — determine if this RTclient is watching the subjects that an RTclient is subscribing to                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientSubscribeGetWatch(srv, client_name, watch_status_return) T_IPC_SRV srv; T_STR client_name; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope)</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Return Values</b> | TRUE if the RTclient subscribe watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientSubscribeGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>T_ERR_NULL_PTR — <i>srv</i>, <i>client_name</i>, or <i>watch_status_return</i> was null</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Description</b>   | <p>TipcSrvMonClientSubscribeGetWatch determines whether this RTclient is watching the subjects that the <i>client_name</i> RTclient is subscribing to. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_SUBSCRIBE_STATUS message each time the watched RTclient starts or stops subscribing to a subject.</p> <p>Each MON_CLIENT_SUBSCRIBE_STATUS message contains four fields:</p> <ul style="list-style-type: none"> <li>a STR field containing the name of the RTclient</li> <li>a STR_ARRAY field containing the names of the current subjects that the RTclient is subscribing to</li> <li>a STR field containing the name of the subject that the RTclient has just started subscribing to (or "" if not applicable)</li> <li>a STR field containing the name of the subject that the RTclient has just stopped subscribing to (or "" if not applicable)</li> </ul> |

To set whether this RTclient is watching the subjects that an RTclient is subscribing to, use `TipcSrvMonClientSubscribeSetWatch`. To poll once for the subjects that an RTclient is subscribing to, use `TipcSrvMonClientSubscribePoll`.

For an example of how `MON_CLIENT_SUBSCRIBE_STATUS` messages can be used, see the RTmon GDI Watch Client Subjects window.

**Caution** If `TipcSrvMonClientSubscribeGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

**See Also** `TipcSrvMonClientSubscribePoll`, `TipcSrvMonClientSubscribeSetWatch`

**Examples** This example uses `TipcSrvMonClientSubscribeGetWatch` to get the status of watching subjects being subscribed to by the primary RTclient:

```
T_BOOL watch_status;

if (!TipcSrvMonClientSubscribeGetWatch(srv, "primary",
&watch_status)) {
 return; /* error */
}
TutOut("This process is %s subjects being subscribed to by ",
 watch_status ? "watching" : "not watching");
TutOut("primary.\n");
```

## TipcSrvMonClientSubscribeNumPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientSubscribeNumPoll — poll for the number of subjects subscribed to by an RTclient                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientSubscribeNumPoll(<i>srv</i>, <i>client_name</i>) T_IPC_SRV <i>srv</i>; T_STR <i>client_name</i>;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Return Values</b> | TRUE if the poll for the number of subjects was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientSubscribeNumPoll fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>client_name</i> was null</li> <li>• any error number from TipcSrvMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b>   | <p>TipcSrvMonClientSubscribeNumPoll polls for the number of subjects subscribed to by an RTclient by sending a MON_CLIENT_SUBSCRIBE_NUM_POLL_CALL message to the RTserver. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The <i>srv</i> argument indicates the connection on which the poll is to be sent.</p> <p>The RTserver responds by sending back a MON_CLIENT_SUBSCRIBE_NUM_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. Each MON_CLIENT_SUBSCRIBE_NUM_POLL_RESULT message contains two fields:</p> <ul style="list-style-type: none"> <li>• a STR_ARRAY field containing the list of client names that match the <i>client_name</i> argument</li> <li>• an INT4_ARRAY field containing the number of subjects subscribed to by each RTclient</li> </ul> |

**Caution** There is no response message if the RTclient named *client\_name* does not exist; the poll is silently dropped.

TipcSrvMonClientSubscribeNumPoll sends a MON\_CLIENT\_SUBSCRIBE\_NUM\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

The number of subjects subscribed to by an RTclient can only be polled, not watched.

**See Also** TipcSrvMonClientSubscribePoll

**Examples** This example uses TipcSrvMonClientSubscribeNumPoll to poll for the number of subjects that all RTclients are subscribing to, waits up to 10 seconds for the poll result, and prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *client_names;
T_INT4 num_client_names;
T_INT4 *num_subject_subscribes;
T_INT4 num_num_subject_subscribes;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientSubscribeNumPoll(srv)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBSCRIBE_NUM_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvConnMsgSearchType(srv, 10.0, mt);
if (msg == NULL) {
 return; /* error */
}
```

```

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
 return; /* error */
}

if (!TipcMsgNextStrArray(msg, &client_names, &num_client_names)
 || !TipcMsgNextInt4Array(msg, &num_subject_subscribes,
 &num_num_subject_subscribes)) {
 return; /* error */
}

TutOut("Got CLIENT_SUBSCRIBE_NUM poll result.\n");
for (counter = 0; counter < num_client_names; counter++) {
 TutOut("client_names[%d] = %s\n", counter,
client_names[counter]);
 TutOut("num_subject_subscribes[%d] = %d\n", counter,
num_subject_subscribes[counter]);
}

/* clean up */
if (!TipcMsgDestroy(msg)) {
 return; /* error */
}

```

## TipcSrvMonClientSubscribePoll

---

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonClientSubscribePoll — poll once for the subjects that an RTclient is subscribing to                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Synopsis      | <pre>T_BOOL TipcSrvMonClientSubscribePoll(srv, client_name) T_IPC_SRV srv; T_STR client_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll for subjects being subscribed to (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Return Values | TRUE if the poll for subjects being subscribed to was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Diagnostics   | <p>If TipcSrvMonClientSubscribePoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> or <i>client_name</i> was null</li><li>• any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Description   | <p>TipcSrvMonClientSubscribePoll polls for subjects that the <i>client_name</i> RTclient is subscribing to by sending a MON_CLIENT_SUBSCRIBE_POLL_CALL message to RTserver. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. RTserver responds by sending back a MON_CLIENT_SUBSCRIBE_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>The MON_CLIENT_SUBSCRIBE_POLL_RESULT message contains one or more groups of two fields:</p> <ul style="list-style-type: none"><li>• a STR field containing the name of the RTclient</li><li>• a STR_ARRAY field containing the names of the current subjects that the RTclient is subscribing to</li></ul> <p>This group of two fields is repeated multiple times only if <i>client_name</i> is a wildcard or T_IPC_MON_ALL.</p> <p>To watch for subjects that an RTclient is subscribing to, use TipcSrvMonClientSubscribeSetWatch.</p> |



**Caution** TipcSrvMonClientSubscribePoll sends a MON\_CLIENT\_SUBSCRIBE\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

**See Also** TipcSrvMonClientSubscribeGetWatch, TipcSrvMonClientSubscribeSetWatch, TipcSrvConnMsgWrite

**Examples** This example uses TipcSrvMonClientSubscribePoll to poll for the subjects that all RTclients are subscribing to, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR client_name;
T_STR *subscribe_subject_names;
T_INT4 num_subscribe_subject_names;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientSubscribePoll(srv, "/...")) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBSCRIBE_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvConnMsgSearchType(srv, 10.0, mt);
if (msg == NULL) {
 return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
 return; /* error */
}

/* loop through all groups */
TutOut("Got CLIENT_SUBSCRIBE poll result.\n");
while (TipcMsgNextStr(msg, &client_name)
 && TipcMsgNextStrArray(msg, &subscribe_subject_names,
 &num_subscribe_subject_names)) {
 TutOut("RTclient name = %s\n", client_name);
 for (counter = 0; counter < num_subscribe_subject_names;
 counter++) {
 TutOut("subscribe_subject_names[%d] = %s\n", counter,
 subscribe_subject_names[counter]);
 }
}
```

```
/* make sure we reached the end of the message */
if (TutErrNumGet() != T_ERR_MSG_EOM) {
 return; /* error */
}

/* clean up */
if (!TpcMsgDestroy(msg)) {
 return; /* error */
}
```

## TipcSrvMonClientSubscribeSetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientSubscribeSetWatch — start or stop watching the subjects that an RTclient is subscribing to                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientSubscribeSetWatch(srv, client_name, watch_status) T_IPC_SRV srv; T_STR client_name; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to set watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope)</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Return Values</b> | TRUE if the RTclient subscribe watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientSubscribeSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>client_name</i> was null</li> <li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li> <li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>   | <p>TipcSrvMonClientSubscribeSetWatch sets whether this RTclient is watching the subjects that the <i>client_name</i> RTclient is subscribing to. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option.</p> <p>TipcSrvMonClientSubscribeSetWatch saves the watch status and sends a MON_CLIENT_SUBSCRIBE_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_CLIENT_SUBSCRIBE_STATUS message each time the watched RTclient starts or stops subscribing to a subject. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.</p> |

Each `MON_CLIENT_SUBSCRIBE_STATUS` message contains four fields:

- a `STR` field containing the name of the RTclient
- a `STR_ARRAY` field containing the names of the current subjects that the RTclient is subscribing to
- a `STR` field containing the name of the subject that the RTclient has just started subscribing to (or "" if not applicable)
- a `STR` field containing the name of the subject that the RTclient has just stopped subscribing to (or "" if not applicable)

In addition to sending a `MON_CLIENT_SUBSCRIBE_STATUS` message each time the watched RTclient starts or stops subscribing to a subject, RTserver also sends an initial `MON_CLIENT_SUBSCRIBE_STATUS` message (or many messages if a wildcard or `T_IPC_MON_ALL` is used) when the watch status is set to `TRUE`, so that the RTclient does not have to wait for the next subscribed subject change to receive the first `MON_CLIENT_SUBSCRIBE_STATUS` message.

To determine if this RTclient is watching the subjects that an RTclient is subscribing to, use `TipcSrvMonClientSubscribeGetWatch`. To poll once for the subjects that an RTclient is subscribing to, use `TipcSrvMonClientSubscribePoll`.

For an example of how `MON_CLIENT_SUBSCRIBE_STATUS` messages can be used, see the RTmon GDI Watch Client Subjects window.

**Caution** `TipcSrvMonClientSubscribeSetWatch` sends a `MON_CLIENT_SUBSCRIBE_SET_WATCH` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no initial status message if the RTclient named *client\_name* does not exist.

**See Also** `TipcSrvMonClientSubscribeGetWatch`, `TipcSrvMonClientSubscribePoll`

**Examples** This example uses TipcSrvMonClientSubscribeSetWatch to start watching the subjects being subscribed to by the primary RTclient and creates a callback to process the incoming MON\_CLIENT\_SUBSCRIBE\_STATUS messages:

```
/* ===== */
/*..process_mon_client_subscribe_status -- process a MON_CLIENT_SUBSCRIBE_STATUS
message */
void T_ENTRY process_mon_client_subscribe_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_STR *subscribe_subject_names;
 T_INT4 num_subscribe_subject_names;
 T_STR start_subject;
 T_STR stop_subject;
 T_INT4 counter;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgNextStr(data->msg, &client_name)
 || !TipcMsgNextStrArray(data->msg, &subscribe_subject_names,
 &num_subscribe_subject_names)
 || !TipcMsgNextStr(data->msg, &start_subject)
 || !TipcMsgNextStr(data->msg, &stop_subject)) {
 return; /* error */
 }

 TutOut("Got CLIENT_SUBSCRIBE status.\n");
 TutOut("RTclient name = %s\n", client_name);
 for (counter = 0; counter < num_subscribe_subject_names;
 counter++) {
 TutOut("subscribe_subject_names[%d] = %s\n", counter,
 subscribe_subject_names[counter]);
 }
 TutOut("start subject = %s\n", start_subject);
 TutOut("stop subject = %s\n", stop_subject);
} /* process_mon_client_subscribe_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonClientSubscribeSetWatch(srv, "primary", TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_SUBSCRIBE_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_subscribe_status,
 NULL) == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```

## TipcSrvMonClientTimeGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientTimeGetWatch — determine if this RTclient is watching time information in an RTclient                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientTimeGetWatch(srv, client_name, watch_status_return) T_IPC_SRV srv; T_STR client_name; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope)</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Return Values</b> | TRUE if the RTclient time information watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientTimeGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>T_ERR_NULL_PTR — <i>srv</i>, <i>client_name</i>, or <i>watch_status_return</i> was null</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>   | <p>TipcSrvMonClientTimeGetWatch determines whether this RTclient is watching time information in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to watch all RTclients in the project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_CLIENT_TIME_STATUS message each time the current time changes in the watched RTclient.</p> <p>Each MON_CLIENT_TIME_STATUS message contains seven fields:</p> <ul style="list-style-type: none"> <li>a STR field containing the name of the RTclient</li> <li>a REAL8 field containing the current time (the time from TutGetCurrentTime)</li> <li>a STR field containing the current time converted to a string using the format specified in the Time_Format option; the time format from the watched RTclient, not this RTclient, is used</li> <li>a REAL8 field containing the wall time (the time from TutGetWallTime)</li> <li>a STR field containing the wall time converted to a string using the full time format</li> </ul> |

- a REAL8 field containing the amount of CPU time used
- an INT4 field containing the number of frames processed

To set whether this RTclient is watching RTclient time information, use `TpcSrvMonClientTimeSetWatch`. To poll once for RTclient time information, use `TpcSrvMonClientTimePoll`.

For an example of how `MON_CLIENT_TIME_STATUS` messages can be used, see the RTmon GDI Watch Client Time window.

**Caution** If `TpcSrvMonClientTimeGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

**See Also** `TpcSrvMonClientTimePoll`, `TpcSrvMonClientTimeSetWatch`

**Examples** This example uses `TpcSrvMonClientTimeGetWatch` to get the status of watching time information in the primary RTclient:

```
T_BOOL watch_status;

if (!TpcSrvMonClientTimeGetWatch(srv, "primary", &watch_status))
{
 return; /* error */
}
TutOut("This process is %s time information in primary.\n",
 watch_status ? "watching" : "not watching");
```



## TipcSrvMonClientTimePoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonClientTimePoll — poll once for RTclient time information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonClientTimePoll(srv, client_name) T_IPC_SRV srv; T_STR client_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll for time information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Return Values</b> | TRUE if the poll for RTclient time information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Diagnostics</b>   | <p>If TipcSrvMonClientTimePoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>client_name</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b>   | <p>TipcSrvMonClientTimePoll polls for RTclient time information by sending a MON_CLIENT_TIME_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_TIME_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_CLIENT_TIME_POLL_RESULT message contains seven fields:</p> <ul style="list-style-type: none"> <li>• a STR field containing the name of the RTclient</li> <li>• a REAL8 field containing the current time (the time from TutGetCurrentTime)</li> <li>• a STR field containing the current time converted to a string using the format specified in the Time_Format option; the time format from the polled RTclient, not this RTclient, is used</li> <li>• a REAL8 field containing the wall time (the time from TutGetWallTime)</li> <li>• a STR field containing the wall time converted to a string using the full time format</li> </ul> |

- a REAL8 field containing the amount of CPU time used
- an INT4 field containing the number of frames processed

There are multiple responses only if *client\_name* is a wildcard or T\_IPC\_MON\_ALL.

To watch RTclient time information, use TipcSrvMonClientTimeSetWatch.

**Caution** TipcSrvMonClientTimePoll sends a MON\_CLIENT\_TIME\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client\_name* does not exist; the poll is silently dropped.

**See Also** TipcSrvMonClientTimeGetWatch, TipcSrvMonClientTimeSetWatch, TipcSrvConnMsgWrite

**Examples** This example uses TipcSrvMonClientTimePoll to poll for time information in all RTclients (including this one), creates a callback to process the incoming MON\_CLIENT\_TIME\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_client_time_poll_result -- process a MON_CLIENT_TIME_POLL_RESULT
message */
void T_ENTRY process_mon_client_time_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_REAL8 current_time;
 T_STR current_time_str;
 T_REAL8 wall_time;
 T_STR wall_time_str;
 T_REAL8 cpu_time;
 T_INT4 frame_count;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }
}
```

```

/* get the fields from the message */
if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_REAL8, &current_time,
 T_IPC_FT_STR, &current_time_str,
 T_IPC_FT_REAL8, &wall_time,
 T_IPC_FT_STR, &wall_time_str,
 T_IPC_FT_REAL8, &cpu_time,
 T_IPC_FT_INT4, &frame_count,
 NULL)) {
 return; /* error */
}

TutOut("Got CLIENT_TIME poll result.\n");
TutOut("RTclient name = %s\n", client_name);
TutOut("current time = %f\n", current_time);
TutOut("current time str = %s\n", current_time_str);
TutOut("wall time = %f\n", wall_time);
TutOut("wall time str = %s\n", wall_time_str);
TutOut("CPU time = %f\n", cpu_time);
TutOut("frame count = %d\n", frame_count);
} /* process_mon_client_time_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonClientTimePoll(srv, "/...")) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_TIME_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_time_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

# TipcSrvMonClientTimeSetWatch

---

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonClientTimeSetWatch — start or stop watching RTclient time information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Synopsis      | <pre>T_BOOL TipcSrvMonClientTimeSetWatch(<i>srv</i>, <i>client_name</i>, <i>watch_status</i>) T_IPC_SRV <i>srv</i>; T_STR <i>client_name</i>; T_BOOL <i>watch_status</i>;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to set watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a watch of all RTclients matching the value of Monitor_Scope)</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Return Values | TRUE if the RTclient time information watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Diagnostics   | <p>If TipcSrvMonClientTimeSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> or <i>client_name</i> was null</li><li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li><li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Description   | TipcSrvMonClientTimeSetWatch sets whether this RTclient is watching time information in the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. TipcSrvMonClientTimeSetWatch saves the watch status and sends a MON_CLIENT_TIME_SET_WATCH message to the <i>client_name</i> RTclient. The watch status controls whether the RTclient receives a MON_CLIENT_TIME_STATUS message each time the current time changes in the watched RTclient. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. |

Each `MON_CLIENT_TIME_STATUS` message contains seven fields:

- a STR field containing the name of the RTclient
- a REAL8 field containing the current time (the time from `TutGetCurrentTime`)
- a STR field containing the current time converted to a string using the format specified in the `Time_Format` option; the time format from the watched RTclient, not this RTclient, is used
- a REAL8 field containing the wall time (the time from `TutGetWallTime`)
- a STR field containing the wall time converted to a string using the full time format
- a REAL8 field containing the amount of CPU time used
- an INT4 field containing the number of frames processed

In addition to sending a `MON_CLIENT_TIME_STATUS` message each time the current time changes, the watched RTclient also sends an initial `MON_CLIENT_TIME_STATUS` message when the watch status is set to `TRUE`, so that the RTclient does not have to wait for the next time change to receive the first `MON_CLIENT_TIME_STATUS` message.

To determine if this RTclient is watching RTclient time information, use `TipcSrvMonClientTimeGetWatch`. To poll once for RTclient time information, use `TipcSrvMonClientTimePoll`.

For an example of how `MON_CLIENT_TIME_STATUS` messages can be used, see the RTmon GDI Watch Client Time window.

**Caution** `TipcSrvMonClientTimeSetWatch` sends a `MON_CLIENT_TIME_SET_WATCH` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no initial status message if the RTclient named *client\_name* does not exist.

**See Also** `TipcSrvMonClientTimeGetWatch`, `TipcSrvMonClientTimePoll`

**Examples** This example uses TpcSrvMonClientTimeSetWatch to start watching time information in the primary RTclient and creates a callback to process the incoming MON\_CLIENT\_TIME\_STATUS messages:

```
/* ===== */
/*..process_mon_client_time_status -- process a MON_CLIENT_TIME_STATUS message */
void T_ENTRY process_mon_client_time_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_REAL8 current_time;
 T_STR current_time_str;
 T_REAL8 wall_time;
 T_STR wall_time_str;
 T_REAL8 cpu_time;
 T_INT4 frame_count;

 /* must set current field first */
 if (!TpcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TpcMsgRead(data->msg,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_REAL8, &current_time,
 T_IPC_FT_STR, &current_time_str,
 T_IPC_FT_REAL8, &wall_time,
 T_IPC_FT_STR, &wall_time_str,
 T_IPC_FT_REAL8, &cpu_time,
 T_IPC_FT_INT4, &frame_count,
 NULL)) {
 return; /* error */
 }

 TutOut("Got CLIENT_TIME status.\n");
 TutOut("RTclient name = %s\n", client_name);
 TutOut("current time = %f\n", current_time);
 TutOut("current time str = %s\n", current_time_str);
 TutOut("wall time = %f\n", wall_time);
 TutOut("wall time str = %s\n", wall_time_str);
 TutOut("CPU time = %f\n", cpu_time);
 TutOut("frame count = %d\n", frame_count);
} /* process_mon_client_time_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonClientTimeSetWatch(srv, "primary", TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_TIME_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_client_time_status,
 NULL)
 == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```

# TipcSrvMonClientVersionPoll

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonClientVersionPoll — poll once for RTclient version information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Synopsis      | <pre>T_BOOL TipcSrvMonClientVersionPoll(srv, client_name) T_IPC_SRV srv; T_STR client_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>client_name</i> — name of RTclient to poll (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTclients matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Return Values | TRUE if the poll was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Diagnostics   | <p>If TipcSrvMonClientVersionPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>T_ERR_NULL_PTR — <i>srv</i> or <i>client_name</i> was null</li><li>any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Description   | <p>TipcSrvMonClientVersionPoll polls for RTclient version information by sending a MON_CLIENT_VERSION_POLL_CALL message to the <i>client_name</i> RTclient. The <i>client_name</i> argument can be the unique subject name of a single RTclient, a wildcarded subject name to match many RTclients, or T_IPC_MON_ALL to poll all RTclients in the project that match the value of the Monitor_Scope option. The polled RTclient responds by sending back a MON_CLIENT_VERSION_POLL_RESULT message. The response may or may not come back quickly, depending on what the polled RTclient is doing. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_CLIENT_VERSION_POLL_RESULT message contains two fields:</p> <ul style="list-style-type: none"><li>a STR field containing the name of the RTclient</li><li>a INT4 field containing the version of the RTclient</li></ul> <p>There are multiple responses only if <i>client_name</i> is a wildcard or T_IPC_MON_ALL.</p> |



**Caution** TipcSrvMonClientVersionPoll sends a MON\_CLIENT\_VERSION\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

There is no response message if the RTclient named *client\_name* does not exist; the poll is silently dropped.

RTclient version information can be polled, but it cannot be watched.

**See Also** TipcMonClientVersionPoll, TipcMonServerVersionPoll

**Examples** This example uses TipcSrvMonClientVersionPoll to poll for version information in all RTclients, creates a callback to process the incoming MON\_CLIENT\_VERSION\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
void T_ENTRY ClientVersionCb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR client_name;
 T_INT4 version;

 TipcMsgNextStr(data->msg, &client_name);
 TipcMsgNextInt4(data->msg, &version);
 TutOut("%s: %d\n", client_name, version);
} /* ClientVersionCb */

/* ... code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_CLIENT_VERSION_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(
 srv, mt, ClientVersionCb, NULL) == NULL) {
 return; /* error */
}

TipcSrvMonClientVersionPoll(srv, "/...");
TipcSrvConnFlush(srv);

if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}
```

# TipcSrvMonExtDelete

---

|               |                                                                                                                                                                                                                                                                      |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonExtDelete — remove a field from an RTclient’s MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                           |
| Synopsis      | <pre>T_BOOL TipcSrvMonExtDelete(srv, field_name) T_IPC_SRV srv; T_STR field_name;</pre>                                                                                                                                                                              |
| Arguments     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message.</p>                                                                                                               |
| Return Values | TRUE if the field was removed, FALSE otherwise.                                                                                                                                                                                                                      |
| Diagnostics   | <p>If TipcSrvMonExtDelete fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li><li>• any error number from TipcMsgDeleteNamedField</li></ul> |
| Description   | TipcSrvMonExtDelete removes a field from an RTclient’s MON_CLIENT_EXT_POLL_RESULT message. The effected MON_CLIENT_EXT_POLL_RESULT message is the one belonging to the RTclient who issued TipcSrvMonExtDelete.                                                      |
| See Also      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                      |
| Examples      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                            |

## TipcSrvMonExtSetBinary

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetBinary — add or update a field of type BINARY in an RTclient's MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                         |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonExtSetBinary(srv, field_name, data, size) T_IPC_SRV srv; T_STR field_name; T_PTR data; T_INT4 size;</pre>                                                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>size</i> — number of bytes in the field.</p>                                                                                                          |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetBinary fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li> <li>• any error number from TipcMsgAddNamedBinary</li> <li>• any error number from TipcMsgUpdateNamedBinary</li> </ul>                                                                                                        |
| <b>Description</b>   | TipcSrvMonExtSetBinary creates a field of type BINARY in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetBinary only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                           |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                 |

# TipcSrvMonExtSetBool

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonExtSetBool — add or update a field of type BOOL in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                       |
| Synopsis      | <pre>T_BOOL TipcSrvMonExtSetBool(<i>srv</i>, <i>field_name</i>, <i>data</i>) T_IPC_SRV <i>srv</i>; T_STR <i>field_name</i>; T_BOOL <i>data</i>;</pre>                                                                                                                                                                                                                                                                               |
| Arguments     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message..</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p>                                                                                                                                                      |
| Return Values | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                            |
| Diagnostics   | <p>If TipcSrvMonExtSetBool fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li><li>• any error number from TipcMsgAddNamedBool</li><li>• any error number from TipcMsgUpdateNamedBool</li></ul>                                                                                                            |
| Description   | TipcSrvMonExtSetBool creates a field of type BOOL in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetBool only updates it with the value of <i>data</i> . |
| See Also      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                     |
| Examples      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                           |

## TipcSrvMonExtSetBoolArray

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetBoolArray — add or update a field containing an array of type BOOL in an RTclient's MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                            |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonExtSetBoolArray(srv, field_name, data, size) T_IPC_SRV srv; T_STR field_name; T_BOOL *data; T_INT4 size;</pre>                                                                                                                                                                                                                                                                                                                              |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>size</i> — number of elements in the field.</p>                                                                                                                               |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetBoolArray fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li> <li>• any error number from TipcMsgAddNamedBoolArray</li> <li>• any error number from TipcMsgUpdateNamedBoolArray</li> </ul>                                                                                                                       |
| <b>Description</b>   | TipcSrvMonExtSetBoolArray creates a field containing an array of type BOOL in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetBoolArray only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                                                   |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                                         |

## TipcSrvMonExtSetInt2

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetInt2 — add or update a field of type INT2 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                       |
| <b>Synopsis</b>      | <pre>T_BOOL  TipcSrvMonExtSetInt2(<i>srv</i>, <i>field_name</i>, <i>data</i>) T_IPC_SRV <i>srv</i>; T_STR   <i>field_name</i>; T_INT2  <i>data</i>;</pre>                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p>                                                                                                                                                       |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetInt2 fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li><li>• any error number from TipcMsgAddNamedInt2</li><li>• any error number from TipcMsgUpdateNamedInt2</li></ul>                                                                                                            |
| <b>Description</b>   | TipcSrvMonExtSetInt2 creates a field of type INT2 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetInt2 only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                     |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                           |

## TipcSrvMonExtSetInt2Array

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetInt2Array — add or update a field containing an array of type INT2 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                            |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonExtSetInt2Array(srv, field_name, data, size) T_IPC_SRV srv; T_STR field_name; T_INT2 *data; T_INT4 size;</pre>                                                                                                                                                                                                                                                                                                                              |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message..</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>size</i> — number of elements in the field.</p>                                                                                                                              |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetInt2Array fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li> <li>• any error number from TipcMsgAddNamedInt2Array</li> <li>• any error number from TipcMsgUpdateNamedInt2Array</li> </ul>                                                                                                                       |
| <b>Description</b>   | TipcSrvMonExtSetInt2Array creates a field containing an array of type INT2 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetInt2Array only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                                                   |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                                         |

## TipcSrvMonExtSetInt4

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetInt4 — add or update a field of type INT4 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                       |
| <b>Synopsis</b>      | <pre>T_BOOL  TipcSrvMonExtSetInt4(<i>srv</i>, <i>field_name</i>, <i>data</i>) T_IPC_SRV <i>srv</i>; T_STR   <i>field_name</i>; T_INT4  <i>data</i>;</pre>                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p>                                                                                                                                                       |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetInt4 fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li><li>• any error number from TipcMsgAddNamedInt4</li><li>• any error number from TipcMsgUpdateNamedInt4</li></ul>                                                                                                            |
| <b>Description</b>   | TipcSrvMonExtSetInt4 creates a field of type INT4 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetInt4 only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                     |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                           |



## TipcSrvMonExtSetInt4Array

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetInt4Array — add or update a field containing an array of type INT4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                            |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonExtSetInt4Array(srv, field_name, data, size) T_IPC_SRV srv; T_STR field_name; T_INT4 *data; T_INT4 size;</pre>                                                                                                                                                                                                                                                                                                                              |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>size</i> — number of elements in the field.</p>                                                                                                                               |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetInt4Array fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li> <li>• any error number from TipcMsgAddNamedInt4Array</li> <li>• any error number from TipcMsgUpdateNamedInt4Array</li> </ul>                                                                                                                       |
| <b>Description</b>   | TipcSrvMonExtSetInt4Array creates a field containing an array of type INT4 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetInt4Array only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                                                   |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                                         |

## TipcSrvMonExtSetInt8

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetInt8 — add or update a field of type INT8 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                       |
| <b>Synopsis</b>      | <pre>T_BOOL  TipcSrvMonExtSetInt8(<i>srv</i>, <i>field_name</i>, <i>data</i>) T_IPC_SRV <i>srv</i>; T_STR  <i>field_name</i>; T_INT8 <i>data</i>;</pre>                                                                                                                                                                                                                                                                             |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p>                                                                                                                                                       |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetInt8 fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li><li>• any error number from TipcMsgAddNamedInt8</li><li>• any error number from TipcMsgUpdateNamedInt8</li></ul>                                                                                                            |
| <b>Description</b>   | TipcSrvMonExtSetInt8 creates a field of type INT8 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetInt8 only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                     |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                           |

## TipcSrvMonExtSetInt8Array

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetInt8Array — add or update a field containing an array of type INT8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                            |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonExtSetInt8Array(<i>srv</i>, <i>field_name</i>, <i>data</i>, <i>size</i>) T_IPC_SRV <i>srv</i>; T_STR <i>field_name</i>; T_INT8 *<i>data</i>; T_INT4 <i>size</i>;</pre>                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>size</i> — number of elements in the field.</p>                                                                                                                               |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetInt8Array fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li> <li>• any error number from TipcMsgAddNamedInt8Array</li> <li>• any error number from TipcMsgUpdateNamedInt8Array</li> </ul>                                                                                                                       |
| <b>Description</b>   | TipcSrvMonExtSetInt8Array creates a field containing an array of type INT8 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetInt8Array only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                                                   |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                                         |

## TipcSrvMonExtSetReal4

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetReal4 — add or update a field of type REAL4 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                        |
| <b>Synopsis</b>      | <pre>T_BOOL  TipcSrvMonExtSetReal4(srv, field_name, data) T_IPC_SRV  srv; T_STR  field_name; T_REAL4  data;</pre>                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message..</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p>                                                                                                                                                         |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetReal4 fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li><li>• any error number from TipcMsgAddNamedReal4</li><li>• any error number from TipcMsgUpdateNamedIntReal4</li></ul>                                                                                                         |
| <b>Description</b>   | TipcSrvMonExtSetReal4 creates a field of type REAL4 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetReal4 only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                        |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                              |

## TipcSrvMonExtSetReal4Array

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetReal4Array — add or update a field containing an array of type REAL4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                             |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonExtSetReal4Array(srv, field_name, data, size) T_IPC_SRV srv; T_STR field_name; T_REAL4 *data; T_INT4 size;</pre>                                                                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>size</i> — number of elements in the field.</p>                                                                                                                                  |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetReal4Array fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li> <li>• any error number from TipcMsgAddNamedReal4Array</li> <li>• any error number from TipcMsgUpdateNamedReal4Array</li> </ul>                                                                                                                       |
| <b>Description</b>   | TipcSrvMonExtSetReal4Array creates a field containing an array of type REAL4 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetReal4Array only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                                                      |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                                            |

## TipcSrvMonExtSetReal8

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetReal8 — adds or updates a field of type REAL8 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                      |
| <b>Synopsis</b>      | <pre>T_BOOL  TipcSrvMonExtSetReal8(srv, field_name, data) T_IPC_SRV  srv; T_STR  field_name; T_REAL8  data;</pre>                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p>                                                                                                                                                          |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetReal8 fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li><li>• any error number from TipcMsgAddNamedReal8</li><li>• any error number from TipcMsgUpdateNamedReal8</li></ul>                                                                                                            |
| <b>Description</b>   | TipcSrvMonExtSetReal8 creates a field of type REAL8 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetReal8 only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                        |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                              |

## TipcSrvMonExtSetReal8Array

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetReal8Array — add or update a field containing an array of type REAL8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                             |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonExtSetReal8Array(srv, field_name, data, size) T_IPC_SRV srv; T_STR field_name; T_REAL8 *data; T_INT4 size;</pre>                                                                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>size</i> — number of elements in the field.</p>                                                                                                                                  |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetReal8Array fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li> <li>• any error number from TipcMsgAddNamedReal8Array</li> <li>• any error number from TipcMsgUpdateNamedReal8Array</li> </ul>                                                                                                                       |
| <b>Description</b>   | TipcSrvMonExtSetReal8Array creates a field containing an array of type REAL8 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetReal8Array only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                                                      |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                                            |

# TipcSrvMonExtSetReal16

|                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name                                                                                | TipcSrvMonExtSetReal16 — add or update a field of type REAL16 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                         |
| Synopsis                                                                            | <pre>T_BOOL TipcSrvMonExtSetReal16(srv, field_name, data) T_IPC_SRV srv; T_STR field_name; T_REAL16 data;</pre>                                                                                                                                                                                                                                                                                                                           |
| Arguments                                                                           | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p>                                                                                                                                                             |
| Return Values                                                                       | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                  |
| Diagnostics                                                                         | <p>If TipcSrvMonExtSetReal16 fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li><li>• any error number from TipcMsgAddNamedReal16</li><li>• any error number from TipcMsgUpdateNamedReal16</li></ul>                                                                                                            |
| Description                                                                         | TipcSrvMonExtSetReal16 creates a field of type REAL16 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetReal16 only updates it with the value of <i>data</i> . |
|  | Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.                                                                                                                                                                                                                                                              |
| See Also                                                                            | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                           |
| Examples                                                                            | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                 |



## TipcSrvMonExtSetReal16Array

---

|                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>                                                                         | TipcSrvMonExtReal16Array — add or update a field containing an array of type REAL16 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                                 |
| <b>Synopsis</b>                                                                     | <pre>T_BOOL TipcSrvMonExtSetReal16Array(srv, field_name, data, size) T_IPC_SRV srv; T_STR field_name; T_REAL16 *data; T_INT4 size;</pre>                                                                                                                                                                                                                                                                                                                                |
| <b>Arguments</b>                                                                    | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>size</i> — number of elements in the field.</p>                                                                                                                                     |
| <b>Return Values</b>                                                                | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Diagnostics</b>                                                                  | <p>If TipcSrvMonExtSetReal16Array fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li> <li>• any error number from TipcMsgAddNamedReal16Array</li> <li>• any error number from TipcMsgUpdateNamedReal16Array</li> </ul>                                                                                                                       |
| <b>Description</b>                                                                  | TipcSrvMonExtSetReal16Array creates a field containing an array of type REAL16 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetReal16Array only updates it with the value of <i>data</i> . |
|  | Not all platforms fully support the REAL16 type. While some platforms can manipulate them as uninterpreted bytes, they can neither manipulate nor interpret them as numbers.                                                                                                                                                                                                                                                                                            |
| <b>See Also</b>                                                                     | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                                                         |
| <b>Examples</b>                                                                     | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                                               |

## TipcSrvMonExtSetStr

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetStr — add or update a field of type STR in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                      |
| <b>Synopsis</b>      | <pre>T_BOOL  TipcSrvMonExtSetStr(<i>srv</i>, <i>field_name</i>, <i>data</i>) T_IPC_SRV <i>srv</i>; T_STR  <i>field_name</i>; T_STR  <i>data</i>;</pre>                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p>                                                                                                                                                    |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetStr fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li><li>• any error number from TipcMsgAddNamedStr</li><li>• any error number from TipcMsgUpdateNamedStr</li></ul>                                                                                                            |
| <b>Description</b>   | TipcSrvMonExtSetStr creates a field of type STR in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetStr only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                  |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                        |

## TipcSrvMonExtSetStrArray

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetStrArray — add or update a field containing an array of type STR in an RTclient's MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                           |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonExtSetStrArray(srv, field_name, data, size) T_IPC_SRV srv; T_STR field_name; T_STR *data; T_INT4 size;</pre>                                                                                                                                                                                                                                                                                                                             |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>size</i> — number of elements in the field.</p>                                                                                                                            |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetStrArray fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li> <li>• any error number from TipcMsgAddNamedStrArray</li> <li>• any error number from TipcMsgUpdateNamedStrArray</li> </ul>                                                                                                                       |
| <b>Description</b>   | TipcSrvMonExtSetStrArray creates a field containing an array of type STR in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetStrArray only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                                                |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                                      |

## TipcSrvMonExtSetUtf8

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetUtf8 — add or update a field of type UTF8 in an RTclient’s MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                       |
| <b>Synopsis</b>      | <pre>T_BOOL  TipcSrvMonExtSetUtf8(<i>srv</i>, <i>field_name</i>, <i>data</i>) T_IPC_SRV <i>srv</i>; T_STR  <i>field_name</i>; T_STR  <i>data</i>;</pre>                                                                                                                                                                                                                                                                             |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p>                                                                                                                                                       |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetUtf8 fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li><li>• any error number from TipcMsgAddNamedUtf8</li><li>• any error number from TipcMsgUpdateNamedUtf8</li></ul>                                                                                                            |
| <b>Description</b>   | TipcSrvMonExtSetUtf8 creates a field of type UTF8 in the RTclient’s MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient’s fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetUtf8 only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                     |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                           |

## TipcSrvMonExtSetUtf8Array

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonExtSetUtf8Array — add or update a field containing an array of type UTF8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message                                                                                                                                                                                                                                                                                                                            |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonExtSetUtf8Array(<i>srv</i>, <i>field_name</i>, <i>data</i>, <i>size</i>) T_IPC_SRV <i>srv</i>; T_STR <i>field_name</i>; T_STR *<i>data</i>; T_INT4 <i>size</i>;</pre>                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to the RTserver</p> <p><i>field_name</i> — name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>data</i> — the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</p> <p><i>size</i> — number of elements in the field.</p>                                                                                                                               |
| <b>Return Values</b> | TRUE if the field was added or updated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Diagnostics</b>   | <p>If TipcSrvMonExtSetUtf8Array fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>field_name</i> was null</li> <li>• any error number from TipcMsgAddNamedUtf8Array</li> <li>• any error number from TipcMsgUpdateNamedUtf8Array</li> </ul>                                                                                                                       |
| <b>Description</b>   | TipcSrvMonExtSetUtf8Array creates a field containing an array of type UTF8 in the RTclient's MON_CLIENT_EXT_POLL_RESULT message to associate with one of the RTclient's fields. If the field in the MON_CLIENT_EXT_POLL_RESULT message does not exist, it is created and updated with the value of <i>data</i> . If the field already exists in the MON_CLIENT_EXT_POLL_RESULT message, TipcSrvMonExtSetUtf8Array only updates it with the value of <i>data</i> . |
| <b>See Also</b>      | Use other TipcSrvMonExt* APIs as necessary to define extension data in each RTclient that is polled by TipcSrvMonClientExtPoll.                                                                                                                                                                                                                                                                                                                                   |
| <b>Examples</b>      | See the example for TipcSrvMonClientExtPoll on page 1251.                                                                                                                                                                                                                                                                                                                                                                                                         |

# TipcSrvMonGetIdentStr

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonGetIdentStr — retrieve the monitoring identification string of this process                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Synopsis      | <pre>T_BOOL TipcSrvMonGetIdentStr(srv, ident_str_return) T_IPC_SRV srv; T_STR *ident_str_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>ident_str_return</i> — storage for identification string</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Return Values | TRUE if the identification string was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Diagnostics   | <p>If TipcSrvMonGetIdentStr fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>T_ERR_NULL_PTR — <i>srv</i> or <i>ident_str_return</i> was null</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description   | <p>TipcSrvMonGetIdentStr gets the monitoring identification string of this process. This identification string is used as a descriptive name for the process when it is being monitored. It is used as part of a field in these message types:</p> <ul style="list-style-type: none"><li>MON_CLIENT_NAMES_STATUS</li><li>MON_SERVER_NAMES_STATUS</li><li>MON_CLIENT_NAMES_POLL_RESULT</li><li>MON_SERVER_NAMES_POLL_RESULT</li></ul> <p>In these message types, the full field has the form "<i>ident</i>: <i>user@node</i>" (for example, "Primary_RTclient: ssuser@workstation1". The monitoring identification string is used as <i>ident</i>. By default the monitoring identification string is based on the most descriptive counted license the process has (for example, Primary_RTclient is more descriptive than RTclient). This string can be set with the function TipcSrvMonSetIdentStr.</p> |
| Caution       | The storage pointed to by <i>ident_str_return</i> should not be modified or freed. The storage is thread safe, but may be overwritten by subsequent calls to SmartSockets APIs within the same thread. If you wish to preserve the value, you should make a copy of the value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| See Also      | TipcSrvMonSetIdentStr                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

**Examples** This example gets and prints the monitoring identification string for this process:

```
T_STR ident_str;

if (!TipcSrvMonGetIdentStr(srv, &ident_str)) {
 return; /* error */
}
TutOut("Monitoring identification string is %s\n", ident_str);
```

# TipcSrvMonPrintWatch

---

|               |                                                                                                                                                                                                              |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonPrintWatch — print all monitoring categories being watched                                                                                                                                         |
| Synopsis      | <pre>T_BOOL TipcSrvMonPrintWatch(<i>srv</i>, <i>func</i>) T_IPC_SRV <i>srv</i>; T_OUT_FUNC <i>func</i>;</pre>                                                                                                |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>func</i> — TutOut-style output function</p>                                                                                                          |
| Return Values | TRUE if the watch information was successfully printed, FALSE otherwise.                                                                                                                                     |
| Diagnostics   | <p>If TipcSrvMonPrintWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>T_ERR_NULL_PTR — <i>srv</i> or <i>func</i> was null</li></ul> |
| Description   | TipcSrvMonPrintWatch prints all the monitoring categories being watched to a TutOut-style output function. The output function is called once for each line of output.                                       |
| Caution       | None                                                                                                                                                                                                         |



**Examples** This example starts watching a few monitoring categories and then calls TipcSrvMonPrintWatch to print out what is being watched:

```
if (!TipcSrvMonSubjectSubscribeSetWatch(srv, T_IPC_MON_ALL, TRUE))
{
 return; /* error */
}
if (!TipcSrvMonProjectNamesSetWatch(srv, TRUE)) {
 return; /* error */
}
if (!TipcSrvMonClientTimeSetWatch(srv, "primary", TRUE)) {
 return; /* error */
}
if (!TipcSrvMonClientMsgSubscribeSetWatch(srv, "primary",
 "numeric_data")) {
 return; /* error */
}
if (!TipcSrvMonPrintWatch(srv, TutOut)) {
 return; /* error */
}
```

The above fragment prints this output:

```
Watching project_names.
Watching subject_subscribe <@>.
Watching client_time <primary>.
Watching client_msg_recv <primary> <numeric_data>.
```

## TipcSrvMonProjectNamesGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonProjectNamesGetWatch — determine wether this RTclient is watching project names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonProjectNamesGetWatch(srv, watch_status_return) T_IPC_SRV srv; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Return Values</b> | TRUE if the project names watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Diagnostics</b>   | <p>If TipcSrvMonProjectNamesGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> or <i>watch_status_return</i> was null</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b>   | <p>TipcSrvMonProjectNamesGetWatch determines whether this RTclient is watching project names. The watch status controls whether the RTclient receives a MON_PROJECT_NAMES_STATUS message each time a project is created or destroyed. A project is considered created when the first RTclient in that project connects to RTserver. A project is considered destroyed when the last RTclient in that project disconnects from RTserver.</p> <p>Each MON_PROJECT_NAMES_STATUS message contains three fields:</p> <ul style="list-style-type: none"><li>• a STR_ARRAY field containing the current project names</li><li>• a STR field containing the name of the created project (or "" if not applicable)</li><li>• a STR field containing the name of the destroyed project (or "" if not applicable)</li></ul> <p>To set whether this RTclient is watching project names, use TipcSrvMonProjectNamesSetWatch. To poll once for project names, use TipcSrvMonProjectNamesPoll.</p> <p>For an example of how MON_PROJECT_NAMES_STATUS messages can be used, see the RTmon GDI Project Browser window.</p> |
| <b>Caution</b>       | If TipcSrvMonProjectNamesGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>See Also</b>      | TipcSrvMonProjectNamesPoll, TipcSrvMonProjectNamesSetWatch                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

**Examples** This example uses TipcSrvMonProjectNamesGetWatch to get the status of watching project names:

```
T_BOOL watch_status;

if (!TipcSrvMonProjectNamesGetWatch(srv, &watch_status)) {
 return; /* error */
}
TutOut("This RTclient is %s project names.\n",
 watch_status ? "watching" : "not watching");
```

# TipcSrvMonProjectNamesPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonProjectNamesPoll — poll once for project names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonProjectNamesPoll(srv) T_IPC_SRV srv;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Arguments</b>     | <i>srv</i> — connection handle to RTserver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Return Values</b> | TRUE if the poll for project names was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Diagnostics</b>   | <p>If TipcSrvMonProjectNamesPoll fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> was null</li><li>• any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                        |
| <b>Description</b>   | <p>TipcSrvMonProjectNamesPoll polls for project names by sending a MON_PROJECT_NAMES_POLL_CALL message to RTserver. RTserver responds by sending back a MON_PROJECT_NAMES_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response. The MON_PROJECT_NAMES_POLL_RESULT message contains one field, a STR_ARRAY field containing the current project names.</p> <p>To watch for created and destroyed project names, use TipcSrvMonProjectNamesSetWatch.</p> |
| <b>Caution</b>       | TipcSrvMonProjectNamesPoll sends a MON_PROJECT_NAMES_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.                                                                                                                                                                                                                                                                                                                                                                        |
| <b>See Also</b>      | TipcSrvMonProjectNamesGetWatch, TipcSrvMonProjectNamesSetWatch, TipcSrvConnMsgWrite                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

**Examples** This example uses TipcSrvMonProjectNamesPoll to poll for project names, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *project_names;
T_INT4 num_project_names;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcSrvMonProjectNamesPoll(srv)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_PROJECT_NAMES_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvConnMsgSearchType(srv, 10.0, mt);
if (msg == NULL) {
 return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
 return; /* error */
}

if (!TipcMsgNextStrArray(msg, &project_names, &num_project_names))
{
 return; /* error */
}

TutOut("Got PROJECT_NAMES poll result.\n");
for (counter = 0; counter < num_project_names; counter++) {
 TutOut("project_names[%d] = %s\n", counter,
project_names[counter]);
}

/* clean up */
if (!TipcMsgDestroy(msg)) {
 return; /* error */
}
```

# TipcSrvMonProjectNamesSetWatch

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonProjectNamesSetWatch — start or stop watching project names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Synopsis      | <pre>T_BOOL TipcSrvMonProjectNamesSetWatch(srv, watch_status) T_IPC_SRV srv; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Return Values | TRUE if the project names watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Diagnostics   | <p>If TipcSrvMonProjectNamesSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> was null</li><li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li><li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Description   | <p>TipcSrvMonProjectNamesSetWatch sets whether this RTclient is watching project names. TipcSrvMonProjectNamesSetWatch saves the watch status and sends a MON_PROJECT_NAMES_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_PROJECT_NAMES_STATUS message each time a project is created or destroyed. A project is considered created when the first RTclient in that project connects to RTserver. A project is considered destroyed when the last RTclient in that project disconnects from RTserver. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.</p> <p>Each MON_PROJECT_NAMES_STATUS message contains three fields:</p> <ul style="list-style-type: none"><li>• a STR_ARRAY field containing the current project names</li><li>• a STR field containing the name of the created project (or "" if not applicable)</li><li>• a STR field containing the name of the destroyed project (or "" if not applicable)</li></ul> <p>In addition to sending a MON_PROJECT_NAMES_STATUS message each time a project is created or destroyed, RTserver also sends an initial MON_PROJECT_NAMES_STATUS message when the watch status is set to TRUE, so that the RTclient does not have to wait for the next project name change to receive the first MON_PROJECT_NAMES_STATUS message.</p> |

To determine if this RTclient is watching project names, use `TipcSrvMonProjectNamesGetWatch`. To poll once for project names, use `TipcSrvMonProjectNamesPoll`.

For an example of how `MON_PROJECT_NAMES_STATUS` messages can be used, see the RTmon GDI Project Browser window.

**Caution** `TipcSrvMonProjectNamesSetWatch` sends a `MON_PROJECT_NAMES_SET_WATCH` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

**See Also** `TipcSrvMonProjectNamesGetWatch`, `TipcSrvMonProjectNamesPoll`

**Examples** This example uses `TipcSrvMonProjectNamesSetWatch` to start watching project names and creates a callback to process the incoming `MON_PROJECT_NAMES_STATUS` messages:

```
/* ===== */
/*..process_mon_project_names_status -- process a MON_PROJECT_NAMES_STATUS message
*/
void T_ENTRY process_mon_project_names_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR *project_names;
 T_INT4 num_project_names;
 T_STR created_project;
 T_STR destroyed_project;
 T_INT4 counter;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgNextStrArray(data->msg, &project_names,
 &num_project_names)
 || !TipcMsgNextStr(data->msg, &created_project)
 || !TipcMsgNextStr(data->msg, &destroyed_project)) {
 return; /* error */
 }

 TutOut("Got PROJECT_NAMES status.\n");
 for (counter = 0; counter < num_project_names; counter++) {
 TutOut("project_names[%d] = %s\n", counter,
 project_names[counter]);
 }
 TutOut("created project = %s\n", created_project);
 TutOut("destroyed project = %s\n", destroyed_project);
} /* process_mon_project_names_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonProjectNamesSetWatch(srv, TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_PROJECT_NAMES_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_project_names_status,
 NULL)
 == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```



# TipcSrvMonServerBufferPoll

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerBufferPoll — poll once for RTserver message-related buffer information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerBufferPoll(srv, server_name, connected_process_name) T_IPC_SRV srv; T_STR server_name; T_STR connected_process_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to poll for buffer information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p> <p><i>connected_process_name</i> — name of connected process whose buffer in RTserver to poll (use T_IPC_MON_ALL to indicate a poll of all processes that the polled RTserver is connected to)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Return Values</b> | TRUE if the poll for RTserver buffer information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerBufferPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>T_ERR_NULL_PTR — <i>srv</i>, <i>server_name</i>, or <i>connected_process_name</i> was null</li> <li>any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Description</b>   | <p>TipcSrvMonServerBufferPoll polls for RTserver message-related buffer information by sending a MON_SERVER_BUFFER_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The buffer information of <i>connected_process_name</i> (which can be either an RTserver or another RTclient in the same project) is polled. The polled RTserver responds by sending back a MON_SERVER_BUFFER_POLL_RESULT message. The response should come back quickly. If <i>connected_process_name</i> is T_IPC_MON_ALL then the buffers in the polled RTserver of all connected processes are polled. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> |

Each `MON_SERVER_BUFFER_POLL_RESULT` message contains one fixed field followed by one or more groups of five fields. The fixed field is a STR field containing the name of the RTserver. The group of five fields is:

- a STR field containing the name of the process RTserver is connected to
- an INT4 field containing the number of queued incoming messages in the RTserver's connection to the connected process
- an INT4 field containing the number of bytes of data in the queued incoming messages in the RTserver's connection to the connected process
- an INT4 field containing the number of bytes of data in the read buffer in the RTserver's connection to the connected process
- an INT4 field containing the number of bytes of data in the write buffer in the RTserver's connection to the connected process

There are multiple responses only if *server\_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTclient buffer information, use `TipcSrvMonClientBufferPoll`.

For an example of how `MON_SERVER_BUFFER_POLL_RESULT` messages can be used, see the RTmon GDI Server Buffers window.

**Caution** `TipcSrvMonServerBufferPoll` sends a `MON_SERVER_BUFFER_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTserver named *server\_name* or connected process named *connected\_process\_name* does not exist; the poll is silently dropped.

RTserver buffer information can only be polled, not watched.

**See Also** `TipcSrvMonClientBufferPoll`, `TipcSrvConnMsgWrite`

**Examples** This example uses TipcSrvMonServerBufferPoll to poll for all buffer information in all RTservers, creates a callback to process the incoming MON\_SERVER\_BUFFER\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```

/* ===== */
/*..process_mon_server_buffer_poll_result -- process a
MON_SERVER_BUFFER_POLL_RESULT message */
void T_ENTRY process_mon_server_buffer_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR server_name;
 T_STR connected_process_name;
 T_INT4 msg_queue_count;
 T_INT4 msg_queue_byte_count;
 T_INT4 read_buffer_count;
 T_INT4 write_buffer_count;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get first fixed field */
 if (!TipcMsgNextStr(data->msg, &server_name)) {
 return; /* error */
 }
 TutOut("Got SERVER_BUFFER poll result.\n");
 TutOut("RTserver name = %s\n", server_name);

 /* loop through all groups */
 while (TipcMsgRead(data->msg,
 T_IPC_FT_STR, &connected_process_name,
 T_IPC_FT_INT4, &msg_queue_count,
 T_IPC_FT_INT4, &msg_queue_byte_count,
 T_IPC_FT_INT4, &read_buffer_count,
 T_IPC_FT_INT4, &write_buffer_count,
 NULL)) {
 TutOut("connected process name = %s\n",
connected_process_name);
 TutOut("msg queue count = %d\n", msg_queue_count);
 TutOut("msg queue byte count = %d\n", msg_queue_byte_count);
 TutOut("read buffer count = %d\n", read_buffer_count);
 TutOut("write buffer count = %d\n", write_buffer_count);
 }

 /* make sure we reached the end of the message */
 if (TutErrNumGet() != T_ERR_MSG_EOM) {
 return; /* error */
 }
} /* process_mon_server_buffer_poll_result */

```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonServerBufferPoll(srv, "...", T_IPC_MON_ALL)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_BUFFER_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_server_buffer_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonServerCongestionGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerCongestionGetWatch — determine if the RTserver is watching for congestion in the RTserver write buffer of a connected process                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerCongestionGetWatch(srv, server_name, connected_process_name, watch_status_return) T_IPC_SRV srv; T_STR server_name; T_STR connected_process_name; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to get watch status for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p> <p><i>connected_process_name</i> — the name of the process to get watch status of (use T_IPC_MON_ALL to indicate a poll of all processes that the polled RTserver is connected to)</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                          |
| <b>Return Values</b> | TRUE if the watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerCongestionGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>T_ERR_NULL_PTR — <i>server_name</i>, <i>connected_process_name</i>, or <i>watch_status_return</i> was null</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Description</b>   | <p>TipcSrvMonServerCongestionGetWatch determines whether the <i>server_name</i> RTserver is watching for congestion in the <i>connected_process_name</i> write buffer. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The <i>connected_process_name</i> argument can be the name of any RTclient connected to <i>server_name</i>, or T_IPC_MON_ALL to indicate a poll of all processes that the polled RTserver is connected to. The <i>srv</i> argument indicates the connection on which the watch is to be checked.</p> <p>The <i>watch_status_return</i> argument is set to TRUE if the watch is currently active, FALSE if not.</p> |

**Caution** TipcSrvMonServerCongestionGetWatch returns FALSE, it does not store a value in *watch\_status\_return*.

Write buffer congestion can only be watched, not polled.

**See Also** TipcSrvMonServerCongestionSetWatch, TipcSrvMonClientCongestionSetWatch, TipcSrvMonClientCongestionGetWatch

**Examples** This example uses TipcSrvMonServerCongestionGetWatch to determine the status of a buffer congestion watch in all connections on all RTservers:

```
T_BOOL watch_status;

if (!TipcSrvMonServerCongestionGetWatch(srv, "/...",
T_IPC_MON_ALL,
 &watch_status)) {
 return; /* error */
}

TutOut("This RTclient is %s RTserver buffer congestion in ",
 watch_status ? "watching" : "not watching");
TutOut("all connections on all servers.\n");
```

## TipcSrvMonServerCongestionSetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerCongestionSetWatch — start or stop watching for congestion in the RTserver write buffer of a connected process                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerCongestionSetWatch(srv, server_name, connected_process_name, high_water, low_water, watch_status) T_IPC_SRV srv; T_STR server_name; T_STR connected_process_name; T_INT4 high_water; T_INT4 low_water; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to set watch status (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p> <p><i>connected_process_name</i> — the name of the process whose write buffer status you want to start or stop watching (use T_IPC_MON_ALL to indicate a poll of all processes that the polled RTserver is connected to)</p> <p><i>high_water</i> — the number of pending bytes that must be in a buffer for it to be considered congested</p> <p><i>low_water</i> — the quantity that the number of pending bytes must fall below to end buffer congestion</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, or FALSE to stop watching)</p> |
| <b>Return Values</b> | TRUE if the watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerCongestionSetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>T_ERR_NULL_PTR — <i>server_name</i> was null</li> <li>T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE, or the value of <i>low_water</i> was less than 0 or greater than the value of <i>high_water</i></li> <li>T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li> </ul>                                                                                                                                                                                                                                                                                                                         |

**Description** TpcSrvMonServerCongestionSetWatch starts or stops the watch of the *connected\_process\_name* write buffer in the *server\_name* RTserver. The *connected\_process\_name* argument can be the name of any RTclient connected to *server\_name*, or T\_IPC\_MON\_ALL to indicate a poll of all processes that the polled RTserver is connected to. The *server\_name* argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T\_IPC\_MON\_ALL to poll all RTservers that match the value of the Monitor\_Scope option. The *srv* argument indicates the connection on which the watch is to be set.

TpcSrvMonServerCongestionSetWatch saves the watch status and sends a MON\_SERVER\_CONGESTION\_SET\_WATCH message to the *server\_name* RTserver. The watch status controls whether the RTclient receives a MON\_SERVER\_CONGESTION\_STATUS message when the number of bytes in the process's write buffer passes the *high\_water* or *low\_water* threshold. When the write buffer reaches *high\_water* messages pending, the RTserver sends a MON\_SERVER\_CONGESTION\_STATUS message. The next MON\_SERVER\_CONGESTION\_STATUS message is sent when the number of bytes in the write buffer reaches the *low\_water* number.

The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.

Each MON\_SERVER\_CONGESTION\_STATUS message contains five fields:

- an INT4 field containing the current size of the write buffer
- an INT4 field containing the threshold value passed (the *high\_water* or *low\_water* number)
- a STR field containing the *server\_name*
- a STR field containing the *connected\_process\_name*
- a BOOL field containing which threshold was passed (TRUE for the *high\_water* threshold, FALSE if the *low\_water* threshold was passed)

Once the number of bytes in the write buffer has reached the *high\_water* number, RTserver does not send another MON\_SERVER\_CONGESTION\_STATUS message until number of pending bytes in the buffer decreases to the *low\_water* number.

**Caution** TpcSrvMonServerCongestionSetWatch sends a MON\_SERVER\_CONGESTION\_SET\_WATCH message to RTserver, but does not explicitly flush the message. See TpcSrvConnGetAutoFlushSize for more information on message buffering.

Write buffer congestion can only be watched, not polled.



**See Also**     TipcSrvMonServerCongestionGetWatch, TipcSrvMonClientCongestionSetWatch, TipcSrvMonClientCongestionGetWatch

**Examples**     This example uses TipcSrvMonServerCongestionSetWatch to start watching buffer congestion in all connection buffers in all RTservers. The *high\_water* threshold is 8 million bytes and the *low\_water* threshold is 2 million bytes. The example creates a callback to process incoming MON\_SERVER\_CONGESTION\_STATUS messages:

```
/* ===== */
/*..process_mon_server_congestion_status – process a
MON_SERVER_CONGESTION_STATUS message */

void process_mon_server_congestion_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_INT4 size;
 T_INT4 threshold;
 T_STR server_name;
 T_STR connected_process_name;
 T_BOOL high_water;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg,
 T_IPC_FT_INT4, &size,
 T_IPC_FT_INT4, &threshold,
 T_IPC_FT_STR, &server_name,
 T_IPC_FT_STR, &connected_process_name,
 T_IPC_FT_BOOL, &high_water,
 NULL)) {
 return; /* error */
 }

 TutOut("Got SERVER_CONGESTION status.\n");
 TutOut("Crossed %s water threshold.\n", high_water ? "high" :
"low");
 TutOut("RTserver name = %s\n", server_name);
 TutOut("connected process name = %s\n", connected_process_name);
 TutOut("buffer size = %d\n", size);
 TutOut("threshold = %d\n", threshold);
} /* process_mon_server_congestion_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonServerCongestionSetWatch (srv, "...",
T_IPC_MON_ALL,
 8000000, 2000000, TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_CONGESTION_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_server_congestion_status,
 NULL) == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```

## TipcSrvMonServerConnGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerConnGetWatch — determine whether this RTclient is watching RTserver connections                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerConnGetWatch(srv, watch_status_return) T_IPC_SRV srv; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Return Values</b> | TRUE if the RTserver connections watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerConnGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>watch_status_return</i> was null</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b>   | <p>TipcSrvMonServerConnGetWatch determines whether this RTclient is watching RTserver connections of the RTservers that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_SERVER_CONN_STATUS message each time a connection between RTservers is created or destroyed.</p> <p>Each MON_SERVER_CONN_STATUS message contains eight fields (all the array fields have the same number of elements):</p> <ul style="list-style-type: none"> <li>• a STR_ARRAY field containing the names of the first RTservers on the connections</li> <li>• a STR_ARRAY field containing the names of the second RTservers on the connections</li> <li>• a STR_ARRAY field containing the logical connection names of the connections</li> <li>• an INT4_ARRAY field containing the weights of the connections (always one)</li> <li>• a STR field containing the name of the first RTserver on the created connection (or "" if not applicable)</li> <li>• a STR field containing the name of the second RTserver on the created connection (or "" if not applicable)</li> </ul> |

- a STR field containing the name of the first RTserver on the destroyed connection (or "" if not applicable)
- a STR field containing the name of the second RTserver on the destroyed connection (or "" if not applicable)

To set whether this RTclient is watching RTserver connections, use `TipcSrvMonServerConnSetWatch`. To poll once for RTserver connections, use `TipcSrvMonServerConnPoll`.

For an example of how `MON_SERVER_CONN_STATUS` messages can be used, see the RTmon GDI Watch Server Connections window.

**Caution** If `TipcSrvMonServerConnGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

**See Also** `TipcSrvMonServerConnPoll`, `TipcSrvMonServerConnSetWatch`

**Examples** This example uses `TipcSrvMonServerConnGetWatch` to get the status of watching RTserver connections:

```
T_BOOL watch_status;

if (!TipcSrvMonServerConnGetWatch(srv, &watch_status)) {
 return; /* error */
}
TutOut("This process is %s RTserver connections.\n",
 watch_status ? "watching" : "not watching");
```

## TipcSrvMonServerConnPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerConnPoll — poll once for RTserver connections                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerConnPoll(srv) T_IPC_SRV srv;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Arguments</b>     | <i>srv</i> — connection handle to RTserver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Return Values</b> | TRUE if the poll for RTserver connections was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerConnPoll fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b>   | <p>TipcSrvMonServerConnPoll polls for RTserver connections of the RTservers that match the value of the Monitor_Scope option by sending a MON_SERVER_CONN_POLL_CALL message to RTserver. RTserver responds by sending back a MON_SERVER_CONN_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>The MON_SERVER_CONN_POLL_RESULT message contains four fields (all the array fields have the same number of elements):</p> <ul style="list-style-type: none"> <li>• a STR_ARRAY field containing the names of the first RTservers on the connections</li> <li>• a STR_ARRAY field containing the names of the second RTservers on the connections</li> <li>• a STR_ARRAY field containing the logical connection names of the connections</li> <li>• an INT4_ARRAY field containing the weights of the connections (always one)</li> </ul> <p>To watch for created and destroyed RTserver connections, use TipcSrvMonServerConnSetWatch.</p> |
| <b>Caution</b>       | TipcSrvMonServerConnPoll sends a MON_SERVER_CONN_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

**See Also** TpcSrvMonServerConnGetWatch, TpcSrvMonServerConnSetWatch, TpcSrvConnMsgWrite

**Examples** This example uses TpcSrvMonServerConnPoll to poll for RTserver connections, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *server1_names;
T_INT4 num_fields;
T_STR *server2_names;
T_STR *conn_names;
T_INT4 *weights;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TpcSrvMonServerConnPoll(srv)) {
 return; /* error */
}

mt = TpcMtLookupByNum(T_MT_MON_SERVER_CONN_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TpcSrvConnMsgSearchType(srv, 10.0, mt);
if (msg == NULL) {
 return; /* error */
}

/* set the current field to the first field */
if (!TpcMsgSetCurrent(msg, 0)) {
 return; /* error */
}

/* all fields contains the same number of elements */
if (!TpcMsgNextStrArray(msg, &server1_names, &num_fields)
 || !TpcMsgNextStrArray(msg, &server2_names, &num_fields)
 || !TpcMsgNextStrArray(msg, &conn_names, &num_fields)
 || !TpcMsgNextInt4Array(msg, &weights, &num_fields)) {
 return; /* error */
}

TutOut("Got SERVER_CONN poll result.\n");
for (counter = 0; counter < num_fields; counter++) {
 TutOut("server1_names[%d] = %s\n", counter,
server1_names[counter]);
 TutOut("server2_names[%d] = %s\n", counter,
server2_names[counter]);
 TutOut("conn_names[%d] = %s\n", counter, conn_names[counter]);
 TutOut("weights[%d] = %d\n", counter, weights[counter]);
}
```

```
/* clean up */
if (!TipcMsgDestroy(msg)) {
 return; /* error */
}
```

# TipcSrvMonServerConnSetWatch

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonServerConnSetWatch — start or stop watching RTserver connections                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Synopsis      | <pre>T_BOOL TipcSrvMonServerConnSetWatch(srv, watch_status) T_IPC_SRV srv; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Return Values | TRUE if the RTserver connections watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Diagnostics   | <p>If TipcSrvMonServerConnSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> was null</li><li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li><li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Description   | <p>TipcSrvMonServerConnSetWatch sets whether this RTclient is watching RTserver connections of the RTservers that match the value of the Monitor_Scope option. TipcSrvMonServerConnSetWatch saves the watch status and sends a MON_SERVER_CONN_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_SERVER_CONN_STATUS message each time a connection between RTservers is created or destroyed. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.</p> <p>Each MON_SERVER_CONN_STATUS message contains eight fields (all the array fields have the same number of elements):</p> <ul style="list-style-type: none"><li>• a STR_ARRAY field containing the names of the first RTservers on the connections</li><li>• a STR_ARRAY field containing the names of the second RTservers on the connections</li><li>• a STR_ARRAY field containing the logical connection names of the connections</li><li>• an INT4_ARRAY field containing the weights of the connections (always one)</li><li>• a STR field containing the name of the first RTserver on the created connection (or "" if not applicable)</li></ul> |



- a STR field containing the name of the second RTserver on the created connection (or "" if not applicable)
- a STR field containing the name of the first RTserver on the destroyed connection (or "" if not applicable)
- a STR field containing the name of the second RTserver on the destroyed connection (or "" if not applicable)

In addition to sending a MON\_SERVER\_CONN\_STATUS message each time an RTserver connection is created or destroyed, RTserver also sends an initial MON\_SERVER\_CONN\_STATUS message when the watch status is set to TRUE, so that the RTclient does not have to wait for the next RTserver connection change to receive the first MON\_SERVER\_CONN\_STATUS message.

To determine if this RTclient is watching RTserver connections, use TipcSrvMonServerConnGetWatch. To poll once for RTserver connections, use TipcSrvMonServerConnPoll.

For an example of how MON\_SERVER\_CONN\_STATUS messages can be used, see the RTmon GDI Watch Server Connections window.

**Caution** TipcSrvMonServerConnSetWatch sends a MON\_SERVER\_CONN\_SET\_WATCH message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

**See Also** TipcSrvMonServerConnGetWatch, TipcSrvMonServerConnPoll

**Examples** This example uses TipcSrvMonServerConnSetWatch to start watching RTserver connections and creates a callback to process the incoming MON\_SERVER\_CONN\_STATUS messages:

```
/* ===== */
/*..process_mon_server_conn_status -- process a MON_SERVER_CONN_STATUS message */
void T_ENTRY process_mon_server_conn_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR *server1_names;
 T_INT4 num_fields;
 T_STR *server2_names;
 T_STR *conn_names;
 T_INT4 *weights;
 T_STR created_server1;
 T_STR created_server2;
 T_STR destroyed_server1;
 T_STR destroyed_server2;
 T_INT4 counter;
```

```

/* must set current field first */
if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
}

/* get the fields from the message */

/* the array fields contain the same number of elements */

if (!TipcMsgNextStrArray(data->msg, &server1_names,
 &num_fields)
 || !TipcMsgNextStrArray(data->msg, &server2_names,
 &num_fields)
 || !TipcMsgNextStrArray(data->msg, &conn_names,
 &num_fields)
 || !TipcMsgNextInt4Array(data->msg, &weights,
 &num_fields)
 || !TipcMsgNextStr(data->msg, &created_server1)
 || !TipcMsgNextStr(data->msg, &created_server2)
 || !TipcMsgNextStr(data->msg, &destroyed_server1)
 || !TipcMsgNextStr(data->msg, &destroyed_server2)) {
 return; /* error */
}

TutOut("Got SERVER_CONN status.\n");
for (counter = 0; counter < num_fields; counter++) {
 TutOut("server1_names[%d] = %s\n", counter,
 server1_names[counter]);
 TutOut("server2_names[%d] = %s\n", counter,
 server2_names[counter]);
 TutOut("conn_names[%d] = %s\n", counter,
 conn_names[counter]);
 TutOut("weights[%d] = %d\n", counter,
 weights[counter]);
}
TutOut("created RTserver 1 = %s\n", created_server1);
TutOut("created RTserver 2 = %s\n", created_server2);
TutOut("destroyed RTserver 1 = %s\n", destroyed_server1);
TutOut("destroyed RTserver 2 = %s\n", destroyed_server2);
} /* process_mon_server_conn_status */

```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonServerConnSetWatch(srv, TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_CONN_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_server_conn_status,
 NULL)
 == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```

# TipcSrvMonServerCpuPoll

---

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonServerCpuPoll — poll for the percentage of CPU time used by an RTserver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Synopsis      | <pre>T_BOOL TipcSrvMonServerCpuPoll(srv, server_name) T_IPC_SRV srv; T_STR server_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to get CPU usage for (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Return Values | TRUE if the poll for the number of subjects was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Diagnostics   | <p>If TipcSrvMonServerCpuPoll fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>T_ERR_NULL_PTR — <i>server_name</i> was null</li><li>any error number from TipcSrvMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Description   | <p>TipcSrvMonServerCpuPoll polls for the percentage of CPU time used by an RTserver by sending a MON_SERVER_CPU_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The <i>srv</i> argument indicates the connection on which the poll is to be sent.</p> <p>The polled RTservers respond by sending back a MON_SERVER_CPU_POLL_RESULT message. Each MON_SERVER_CPU_POLL_RESULT contains two fields:</p> <ul style="list-style-type: none"><li>a STR field containing the RTserver name</li><li>a REAL4 field containing the percentage of CPU time used (between 0.0 and 100.0)</li></ul> <p>There are multiple responses only if <i>server_name</i> is a wildcard or T_IPC_MON_ALL.</p> <p>If the RTserver cannot query the operating system, the value reported in the CPU percentage used field is a negative number.</p> |

**Caution** There is no response message if the RTserver named *server\_name* does not exist; the poll is silently dropped.

TipcSrvMonServerCpuPoll sends a MON\_SERVER\_CPU\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

RTserver CPU usage information can only be polled, not watched.

**See Also** TipcSrvMonClientCpuPoll

**Examples** This example uses TipcSrvMonServerCpuPoll to poll for information in all RTservers, creates a callback to process the incoming MON\_SERVER\_CPU\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results.:

```
/* ===== */
/*..process_mon_server_cpu_poll_result - process a
MON_SERVER_CPU_POLL_RESULT message */

void process_mon_server_cpu_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR server_name;
 T_REAL4 cpu_utilization;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &server_name,
 T_IPC_FT_REAL4, &cpu_utilization,
 NULL)) {
 return; /* error */
 }

 TutOut("Got SERVER_CPU poll result.\n");
 TutOut("RTserver name = %s\n", server_name);
 TutOut("CPU utilization percentage = %f\n", cpu_utilization);
} /* process_mon_server_cpu_poll_result */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TpcSrvMonServerCpuPoll(srv, "...")) {
 return; /* error */
}

mt = TpcMtLookupByNum(T_MT_MON_SERVER_CPU_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TpcSrvConnProcessCbCreate(srv, mt,
 process_mon_server_cpu_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming messages) that arrive within the
next 10 seconds */

if (!TpcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonServerGeneralPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerGeneralPoll — poll once for RTserver general information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerGeneralPoll(srv, server_name) T_IPC_SRV srv; T_STR server_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to poll for general information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Return Values</b> | TRUE if the poll for RTserver general information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerGeneralPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>server_name</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>   | <p>TipcSrvMonServerGeneralPoll polls for RTserver general information by sending a MON_SERVER_GENERAL_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTserver responds by sending back a MON_SERVER_GENERAL_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_SERVER_GENERAL_POLL_RESULT message contains seventeen fields:</p> <ul style="list-style-type: none"> <li>• a STR field containing the name of the RTserver</li> <li>• a STR field containing the process monitoring identifier (defaults to a major type of license, such as RTserver)</li> <li>• a STR field containing the node name the RTserver is running on</li> <li>• a STR field containing the user name the RTserver is running under</li> <li>• an INT4 field containing the PID of the RTserver</li> </ul> |

- a STR field containing the name of the SmartSockets architecture (such as `sun4_solaris`)
- an INT4 field containing the current operating system `sbrk` value (the edge of the RTserver's virtual memory address space)
- an INT4 field containing the amount of change in the current operating system `sbrk` value (the edge of the process's virtual memory address space) since the RTserver started (actually, since the process's first call to any SmartSockets function)
- an INT2 field containing the integer number format of the RTserver (really of type `T_INT_FORMAT`, but stored in an INT2 field to save space)
- an INT2 field containing the real number format of the RTserver (really of type `T_REAL_FORMAT`, but stored in an INT2 field to save space)
- a STR field containing the name of the startup command file that the RTserver used at initialization
- an INT4 field containing whether the RTserver was started with the `-no_daemon` command-line argument (really of type `BOOL`)
- an INT4 field containing whether the RTserver was started with the `-cmd_mode` command-line argument (really of type `BOOL`)
- a STR\_ARRAY field containing the names of the RTclients in the same project as the polling RTclient which are directly connected to the RTserver
- a STR\_ARRAY field containing the names of the RTservers which are directly connected to the RTserver
- a STR\_ARRAY field containing the names of the subjects in the same project as the polling RTclient to which the RTserver is subscribing as the result of an RTserver `subscribe` command
- a STR\_ARRAY field containing the names of the subjects in the same project as the polling RTclient to which the RTserver is subscribing as the result of direct RTclient subscriptions

There are multiple responses only if *server\_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTclient general information, use `TipcSrvMonClientGeneralPoll`.

For an example of how `MON_SERVER_GENERAL_POLL_RESULT` messages can be used, see the RTmon GDI Server Information window.



**Caution** TipcSrvMonServerGeneralPoll sends a MON\_SERVER\_GENERAL\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

There is no response message if the RTserver named *server\_name* does not exist; the poll is silently dropped.

RTserver general information can only be polled, not watched.

**See Also** TipcSrvMonClientGeneralPoll, TipcSrvConnMsgWrite

**Examples** This example uses TipcSrvMonServerGeneralPoll to poll for general information in all RTservers, creates a callback to process the incoming MON\_SERVER\_GENERAL\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_general_poll_result -- process a
MON_SERVER_GENERAL_POLL_RESULT message */
void T_ENTRY process_mon_server_general_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR server_name;
 T_STR ident;
 T_STR node_name;
 T_STR user_name;
 T_INT4 pid;
 T_STR arch;
 T_INT4 current_sbrk;
 T_INT4 delta_sbrk;
 T_INT2 int_format;
 T_INT2 real_format;
 T_STR cmd_file_name;
 T_INT4 no_daemon_flag;
 T_INT4 cmd_mode_flag;
 T_STR *direct_client_names;
 T_INT4 num_direct_client_names;
 T_STR *direct_server_names;
 T_INT4 num_direct_server_names;
 T_STR *server_subscribes;
 T_INT4 num_server_subscribes;
 T_STR *client_subscribes;
 T_INT4 num_client_subscribes;
 T_INT4 counter;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }
}
```

```

/* get the fields from the message */
if (!TpcMsgRead(data->msg,
 T_IPC_FT_STR, &server_name,
 T_IPC_FT_STR, &ident,
 T_IPC_FT_STR, &node_name,
 T_IPC_FT_STR, &user_name,
 T_IPC_FT_INT4, &pid,
 T_IPC_FT_STR, &arch,
 T_IPC_FT_INT4, &current_sbrk,
 T_IPC_FT_INT4, &delta_sbrk,
 T_IPC_FT_INT2, &int_format,
 T_IPC_FT_INT2, &real_format,
 T_IPC_FT_STR, &cmd_file_name,
 T_IPC_FT_INT4, &no_daemon_flag,
 T_IPC_FT_INT4, &cmd_mode_flag,
 T_IPC_FT_STR_ARRAY, &direct_client_names,
 &num_direct_client_names,
 T_IPC_FT_STR_ARRAY, &direct_server_names,
 &num_direct_server_names,
 T_IPC_FT_STR_ARRAY, &server_subscribes,
 &num_server_subscribes,
 T_IPC_FT_STR_ARRAY, &client_subscribes,
 &num_client_subscribes,
 NULL)) {
 return; /* error */
}

TutOut("Got SERVER_GENERAL poll result.\n");
TutOut("RTserver name = %s\n", server_name);
TutOut("ident = %s\n", ident);
TutOut("node name = %s\n", node_name);
TutOut("user name = %s\n", user_name);
TutOut("pid = %d\n", pid);
TutOut("architecture = %s\n", arch);
TutOut("current sbrk value = %d\n", current_sbrk);
TutOut("delta sbrk value = %d\n", delta_sbrk);
TutOut("int format = %d\n", int_format);
TutOut("real format = %d\n", real_format);
TutOut("cmd file name = %s\n", cmd_file_name);
TutOut("-no_daemon flag = %s\n", no_daemon_flag ? "TRUE" :
"FALSE");
TutOut("-cmd_mode flag = %s\n", cmd_mode_flag ? "TRUE" :
"FALSE");

TutOut(" direct client names:");
for (counter = 0; counter < num_direct_client_names; counter++) {
 TutOut(" %s", direct_client_names[counter]);
}
TutOut("\n");

TutOut(" direct server names:");
for (counter = 0; counter < num_direct_server_names; counter++) {
 TutOut(" %s", direct_server_names[counter]);
}
TutOut("\n");

```

```

 TutOut(" server subscribes:");
 for (counter = 0; counter < num_server_subscribes; counter++) {
 TutOut(" %s", server_subscribes[counter]);
 }
 TutOut("\n");
 TutOut(" client subscribes:");
 for (counter = 0; counter < num_client_subscribes; counter++) {
 TutOut(" %s", client_subscribes[counter]);
 }
 TutOut("\n");
} /*process_mon_server_general_poll_result */

/*===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonServerGeneralPoll(srv, "...")) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_GENERAL_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}
if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_server_general_poll_result,
 NULL) == NULL) {
 return; /* error */
}
/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonServerMaxClientLicensesGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerMaxClientLicensesGetWatch — determine if the RTserver is watching for the RTserver cloud to exceed the licensed number of RTclient connections                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerMaxClientLicensesGetWatch(srv, server_name, watch_status_return) T_IPC_SRV srv; T_STR server_name; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to set watch status (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                   |
| <b>Return Values</b> | TRUE if the watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerMaxClientLicensesGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>server_name</i> or <i>watch_status_return</i> was null</li></ul>                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b>   | <p>TipcSrvMonServerMaxClientLicensesGetWatch determines whether the <i>server_name</i> RTserver is watching for the server cloud to exceed the maximum licensed number of RTclient connections. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The <i>srv</i> argument indicates the connection on which the watch is to be checked.</p> <p>The <i>watch_status_return</i> argument is set to TRUE if the watch is currently active, FALSE if not.</p> |
| <b>Caution</b>       | <p>TipcSrvMonServerCongestionGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i>.</p> <p>RTserver licenses can only be watched, not polled.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>See Also</b>      | TipcSrvMonServerMaxClientLicensesSetWatch                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

**Examples** This example uses TipcSrvMonServerMaxClientLicensesGetWatch to determine whether the RTserver is watching for the RTserver cloud to exceed the maximum licensed number of RTclient connections:

```
T_BOOL watch_status;

if (!TipcSrvMonServerMaxClientLicensesGetWatch(srv, "...",
 &watch_status)) {
 return; /* error */
}

TutOut("This RTclient is %s client rejections in ",
 watch_status ? "watching" : "not watching");
TutOut("all servers.\n");
```

# TipcSrvMonServerMaxClientLicensesSetWatch

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonServerMaxClientLicensesSetWatch — start or stop watching for the RTserver cloud to exceed the licensed number of RTclient connections                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Synopsis      | <pre>T_BOOL TipcSrvMonServerMaxClientLicensesSetWatch(srv, server_name, watch_status) T_IPC_SRV srv; T_STR server_name; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to set watch status (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Return Values | TRUE if the watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Diagnostics   | <p>If TipcSrvMonServerMaxClientLicensesSetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>server_name</i> was null</li><li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li><li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description   | <p>TipcSrvMonServerMaxClientLicensesSetWatch sets whether the RTserver is watching for the RTserver cloud to exceed the maximum licensed number of RTclient connections. TipcSrvMonServerMaxClientLicensesSetWatch saves the watch status and sends a MON_SERVER_MAX_CLIENT_LICENSES_SET_WATCH message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The <i>srv</i> argument indicates the connection on which the watch is to be set.</p> <p>The watch status controls whether the RTclient receives a MON_SERVER_MAX_CLIENT_LICENSES_STATUS message each time the RTserver exceeds the allowed number of client connections and refuses a new RTclient connection. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.</p> |

Each MON\_SERVER\_MAX\_CLIENT\_LICENSES\_STATUS contains three fields:

- an INT4 field containing the number of RTclient connections licensed
- a STR field containing the name of the RTclient that was refused a connection
- a STR field containing the name of the RTserver that refused the connection

**Caution** TipcSrvMonServerMaxClientLicensesSetWatch sends a MON\_SERVER\_MAX\_CLIENT\_LICENSES\_SET\_WATCH message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

RTserver licenses can only be watched, not polled.

**See Also** TipcSrvMonServerMaxClientLicensesGetWatch

**Examples** This example uses TipcSrvMonServerMaxClientLicensesSetWatch to start watching all RTservers for client connections that were rejected because the maximum number of client licenses is exceeded, and creates a callback to process the incoming MON\_SERVER\_MAX\_CLIENT\_LICENSES\_STATUS messages:

```
/* ===== */
/*..process_mon_server_max_client_licenses_status – process a
MON_SERVER_MAX_CLIENT_LICENSESESE_STATUS message */

void process_mon_server_max_client_licenses_status(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_INT4 max_licenses;
 T_STR client_name;
 T_STR server_name;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg,
 T_IPC_FT_INT4, &max_licenses,
 T_IPC_FT_STR, &client_name,
 T_IPC_FT_STR, &server_name,
 NULL)) {
 return; /* error */
 }

 TutOut("Got SERVER_MAX_CLIENT_LICENSES status.\n");
 TutOut("rejected RTclient name = %s\n", client_name);
 TutOut("rejecting RTserver name = %s\n", server_name);
} /* process_mon_server_max_client_licenses_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonServerMaxClientLicensesSetWatch (srv, "...",
TRUE))
 return; /* error */
}

mt =
TipcMtLookupByNum(T_MT_MON_SERVER_MAX_CLIENT_LICENSES_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
process_mon_server_max_client_licenses_status,
NULL) == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```



## TipcSrvMonServerMsgTrafficExPoll

---

**Name**      TipcSrvMonServerMsgTrafficExPoll — poll once for RTserver message traffic information



This new function supersedes `TipcSrvMonServerMsgTrafficPoll` for clients that use release 6.7 or later.

**Synopsis**      `T_BOOL TipcSrvMonServerMsgTrafficExPoll(srv, server_name, connected_process_name)`  
                  `T_IPC_SRV srv;`  
                  `T_STR server_name;`  
                  `T_STR connected_process_name;`

**Arguments**      `srv` — connection handle to RTserver

`server_name` — name of RTserver to poll for message traffic information (wildcard names allowed, or use `T_IPC_MON_ALL` to indicate a poll of all RTservers matching the value of `Monitor_Scope`)

`connected_process_name` — name of connected process whose message traffic in RTserver to poll (use `T_IPC_MON_ALL` to indicate a poll of all processes that the polled RTserver is connected to)

**Return Values**      `TRUE` if the poll for RTserver message traffic information was successfully initiated, `FALSE` otherwise.

**Diagnostics**      If `TipcSrvMonServerMsgTrafficExPoll` fails, it returns `FALSE` and sets the global SmartSockets error number to one of:

- `T_ERR_NULL_PTR` — `srv`, `server_name` or `connected_process_name` was null
- any error number from `TipcSrvConnMsgWrite`

**Description**      `TipcSrvMonServerMsgTrafficExPoll` polls for RTserver message traffic information by sending a `MON_SERVER_MSG_TRAFFIC_EX_POLL_CALL` message to the `server_name` RTserver. The `server_name` argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or `T_IPC_MON_ALL` to poll all RTservers that match the value of the `Monitor_Scope` option. The message traffic information of `connected_process_name` (which can be either an RTserver or another RTclient in the same project) is polled. The polled RTserver responds by sending back a `MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT` message. The response

should come back quickly. If *connected\_process\_name* is `T_IPC_MON_ALL` then the message traffic in the polled RTserver of all connected processes are polled. The polling RTclient can use functions such as `TpcSrvMsgSearchType` or `TpcSrvMainLoop` to get the response.

Each `MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT` message contains one fixed field followed by five array fields. The fixed field is a STR field containing the name of the RTserver.

The array fields are:

- a `STR_ARRAY` field containing the name of the process that the RTserver is connected to
- an `INT8_ARRAY` field containing the number of messages the polled RTserver has sent on its connection to each connected process
- an `INT8_ARRAY` field containing the number of messages the polled RTserver has received on its connection to each connected process
- an `INT8_ARRAY` field containing the number of bytes the polled RTserver has sent on its connection to each connected process
- an `INT8_ARRAY` field containing the number of bytes the polled RTserver has received on its connection to each connected process

There are multiple responses only if *server\_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTclient message traffic information, use `TpcSrvMonClientMsgTrafficPoll`.

For an example of how `MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT` messages can be used, see the RTmon GDI Watch Server Connections window.

**Caution** `TpcSrvMonServerMsgTrafficPoll` sends a `MON_SERVER_MSG_TRAFFIC_EX_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TpcSrvGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTserver named *server\_name* does not exist; the poll is silently dropped.

RTserver message traffic information can only be polled, not watched.

**See Also** `TpcSrvMonClientMsgTrafficPoll`, `TpcSrvConnMsgWrite`

**Examples** This example uses TipcSrvMonServerMsgTrafficExPoll to poll for all message traffic information in all RTservers, creates a callback to process the incoming MON\_SERVER\_MSG\_TRAFFIC\_EX\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_msg_traffic_ex_poll_result -- process a
MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT message */

void process_mon_server_msg_traffic_ex_poll_result
(
 T_IPC_CONN conn,
 T_IPC_CONN_PROCESS_CB_DATA data,
 T_CB_ARG arg
)
{
 T_MSG msg;
 T_STR server_name;
 T_STR *process_name;
 T_INT4 num_processes;
 T_INT8 *total_msg_send;
 T_INT4 num_msg_send;
 T_INT8 *total_msg_recv;
 T_INT4 num_msg_recv;
 T_INT8 *total_byte_send;
 T_INT4 num_byte_send;
 T_INT8 *total_byte_recv;
 T_INT4 num_byte_recv;
 T_INT4 counter;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 msg = data->msg;
 if (!TipcMsgNextStr(msg, &server_name)
 || !TipcMsgNextStrArray(msg, &process_name, &num_processes)
 || !TipcMsgNextInt8Array(msg, &total_msg_send, &num_msg_send)
 || !TipcMsgNextInt8Array(msg, &total_msg_recv, &num_msg_recv)
 || !TipcMsgNextInt8Array(msg, &total_byte_send, &num_byte_send)
 || !TipcMsgNextInt8Array(msg, &total_byte_recv, &num_byte_recv))
 {
 return; return; /* error */
 }
 TutOut("Got SERVER_MSG_TRAFFIC_EX poll result.\n");
 TutOut("RTserver name = %s\n", server_name);
}
```

```

/* loop through all groups */
for (counter=0; counter<num_processes; counter++) {
 TutOut("connected process name = %s\n", process_name[counter]);
/*
* IMPORTANT: The format string for T_INT8 differs by platform.
* The T_INT8_SPEC utility macro supplies the correct format
* string for each platform.
*/
 TutOut("total msg send = " T_INT8_SPEC "\n",
 total_msg_send[counter]);
 TutOut("total msg recv = " T_INT8_SPEC "\n",
 total_msg_recv[counter]);
 TutOut("total byte send = " T_INT8_SPEC "\n",
 total_byte_send[counter]);
 TutOut("total byte recv = " T_INT8_SPEC "\n",
 total_byte_recv[counter]);
}

}/* process_mon_server_msg_traffic_ex_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

mt =
TipcMtLookupByNum(T_MT_MON_SERVER_MSG_TRAFFIC_EX_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}
if (TipcSrvConnProcessCbCreate(srv, mt,

process_mon_server_msg_traffic_ex_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* send the poll request out to RTserver */
if (!TipcSrvMonServerMsgTrafficExPoll(srv, "/...", T_IPC_MON_ALL))
{
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonServerMsgTrafficPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerMsgTrafficPoll — poll once for RTserver message traffic information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerMsgTrafficPoll(srv, server_name, connected_process_name) T_IPC_SRV srv; T_STR server_name; T_STR connected_process_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to poll for message traffic information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p> <p><i>connected_process_name</i> — name of connected process whose message traffic in RTserver to poll (use T_IPC_MON_ALL to indicate a poll of all processes that the polled RTserver is connected to)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Return Values</b> | TRUE if the poll for RTserver message traffic information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerMsgTrafficPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i>, <i>server_name</i>, or <i>connected_process_name</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b>   | <p>TipcSrvMonServerMsgTrafficPoll polls for RTserver message traffic information by sending a MON_SERVER_MSG_TRAFFIC_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The message traffic information of <i>connected_process_name</i> (which can be either an RTserver or another RTclient in the same project) is polled. The polled RTserver responds by sending back a MON_SERVER_MSG_TRAFFIC_POLL_RESULT message. The response should come back quickly. If <i>connected_process_name</i> is T_IPC_MON_ALL then the message traffic in the polled RTserver of all connected processes is polled. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> |

Each `MON_SERVER_MSG_TRAFFIC_POLL_RESULT` message contains one fixed field followed by one or more groups of five fields. The fixed field is a STR field containing the name of the RTserver. The group of five fields is:

- a STR field containing the name of the process RTserver is connected to
- an INT4 field containing the number of messages the polled RTserver has sent on its connection to the connected process
- an INT4 field containing the number of messages the polled RTserver has received on its connection to the connected process
- an INT4 field containing the number of bytes the polled RTserver has sent on its connection to the connected process
- an INT4 field containing the number of bytes the polled RTserver has received on its connection to the connected process

There are multiple responses only if *server\_name* is a wildcard or `T_IPC_MON_ALL`.

To poll once for RTclient message traffic information, use `TipcSrvMonClientMsgTrafficPoll`.

For an example of how `MON_SERVER_MSG_TRAFFIC_POLL_RESULT` messages can be used, see the RTmon GDI Watch Server Connections window.

**Caution** `TipcSrvMonServerMsgTrafficPoll` sends a `MON_SERVER_MSG_TRAFFIC_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTserver named *server\_name* does not exist; the poll is silently dropped.

RTserver message traffic information can only be polled, not watched.

**See Also** `TipcSrvMonClientMsgTrafficPoll`, `TipcSrvConnMsgWrite`

**Examples** This example uses TipcSrvMonServerMsgTrafficPoll to poll for all message traffic information in all RTservers, creates a callback to process the incoming MON\_SERVER\_MSG\_TRAFFIC\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_msg_traffic_poll_result -- process a
MON_SERVER_MSG_TRAFFIC_POLL_RESULT message */
void T_ENTRY process_mon_server_msg_traffic_poll_result(conn,
data,
arg)

T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR server_name;
 T_STR connected_process_name;
 T_INT4 total_msg_send;
 T_INT4 total_msg_rcv;
 T_INT4 total_byte_send;
 T_INT4 total_byte_rcv;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get first fixed field */
 if (!TipcMsgNextStr(data->msg, &server_name)) {
 return; /* error */
 }
 TutOut("Got SERVER_MSG_TRAFFIC poll result.\n");
 TutOut("RTserver name = %s\n", server_name);

 /* loop through all groups */
 while (TipcMsgRead(data->msg,
 T_IPC_FT_STR, &connected_process_name,
 T_IPC_FT_INT4, &total_msg_send,
 T_IPC_FT_INT4, &total_msg_rcv,
 T_IPC_FT_INT4, &total_byte_send,
 T_IPC_FT_INT4, &total_byte_rcv,
 NULL)) {
 TutOut("connected process name = %s\n",
connected_process_name);
 TutOut("total msg send = %d\n", total_msg_send);
 TutOut("total msg rcv = %d\n", total_msg_rcv);
 TutOut("total byte send = %d\n", total_byte_send);
 TutOut("total byte rcv = %d\n", total_byte_rcv);
 }

 /* make sure we reached the end of the message */
 if (TutErrNumGet() != T_ERR_MSG_EOM) {
 return; /* error */
 }
}
```

```

} /* process_mon_server_msg_traffic_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonServerMsgTrafficPoll(srv, "...", T_IPC_MON_ALL)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_MSG_TRAFFIC_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}
if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_server_msg_traffic_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```



## TipcSrvMonServerNamesGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerNamesGetWatch — determine if this RTclient is watching RTserver names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerNamesGetWatch(srv, watch_status_return) T_IPC_SRV srv; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Return Values</b> | TRUE if the RTserver names watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerNamesGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>T_ERR_NULL_PTR — <i>srv</i> or <i>watch_status_return</i> was null</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Description</b>   | <p>TipcSrvMonServerNamesGetWatch determines whether this RTclient is watching the RTserver names that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_SERVER_NAMES_STATUS message each time an RTserver is created or destroyed. An RTserver is considered created when it starts up (and usually connects to other RTservers). An RTserver is considered destroyed when it terminates or disconnects from other RTservers.</p> <p>Each MON_SERVER_NAMES_STATUS message contains four fields:</p> <ul style="list-style-type: none"> <li>a STR_ARRAY field containing the current RTserver names</li> <li>a STR_ARRAY field containing RTserver info strings of the form "<i>ident: user@node</i>" (for example, "Primary_RTclient: ssuser@workstation1")</li> <li>a STR field containing the name of the created RTserver (or "" if not applicable)</li> <li>a STR field containing the name of the destroyed RTserver (or "" if not applicable)</li> </ul> <p>To set whether this RTclient is watching RTserver names, use TipcSrvMonServerNamesSetWatch. To poll once for RTserver names, use TipcSrvMonServerNamesPoll.</p> |
| <b>Caution</b>       | If TipcSrvMonServerNamesGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

**See Also**     TpcSrvMonServerNamesPoll, TpcSrvMonServerNamesSetWatch

**Examples**     This example uses TpcSrvMonServerNamesGetWatch to get the status of watching RTserver names:

```
T_BOOL watch_status;

if (!TpcSrvMonServerNamesGetWatch(srv, &watch_status)) {
 return; /* error */
}
TutOut("This process is %s RTserver names.\n",
 watch_status ? "watching" : "not watching");
```

## TipcSrvMonServerNamesPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerNamesPoll — poll once for RTserver names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerNamesPoll(srv) T_IPC_SRV srv;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Arguments</b>     | <i>srv</i> — connection handle to RTserver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Return Values</b> | TRUE if the poll for RTserver names was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerNamesPoll fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Description</b>   | <p>TipcSrvMonServerNamesPoll polls for the RTserver names that match the value of the Monitor_Scope option by sending a MON_SERVER_NAMES_POLL_CALL message to RTserver. RTserver responds by sending back a MON_SERVER_NAMES_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response. The MON_SERVER_NAMES_POLL_RESULT message contains two fields:</p> <ul style="list-style-type: none"> <li>• a STR_ARRAY field containing the current RTserver names</li> <li>• a STR_ARRAY field containing RTserver info strings of the form "<i>ident: user@node</i>" (for example, "RTserver: ssuser@workstation1")</li> </ul> <p>To watch for created and destroyed RTserver names, use TipcSrvMonServerNamesSetWatch.</p> |
| <b>Caution</b>       | TipcSrvMonServerNamesPoll sends a MON_SERVER_NAMES_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>See Also</b>      | TipcSrvMonServerNamesGetWatch, TipcSrvMonServerNamesSetWatch, TipcSrvConnMsgWrite                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

**Examples** This example uses TipcSrvMonServerNamesPoll to poll for RTserver names, waits up to 10 seconds for the poll result, and then prints the poll result message:

```

T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *server_names;
T_INT4 num_server_names;
T_STR *info_strs;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcSrvMonServerNamesPoll(srv)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_NAMES_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvConnMsgSearchType(srv, 10.0, mt);
if (msg == NULL) {
 return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
 return; /* error */
}

/* both server_names and info_strs contains the same number of elements */
if (!TipcMsgNextStrArray(msg, &server_names, &num_server_names)
 || !TipcMsgNextStrArray(msg, &info_strs, &num_server_names)) {
 return; /* error */
}

TutOut("Got SERVER_NAMES poll result.\n");
for (counter = 0; counter < num_server_names; counter++) {
 TutOut("server_names[%d] = %s\n", counter,
server_names[counter]);
 TutOut("info_strs[%d] = %s\n", counter, info_strs[counter]);
}

/* clean up */
if (!TipcMsgDestroy(msg)) {
 return; /* error */
}

```

## TipcSrvMonServerNamesSetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerNamesSetWatch — start or stop watching RTserver names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerNamesSetWatch(srv, watch_status) T_IPC_SRV srv; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Return Values</b> | TRUE if the RTserver names watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerNamesSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> was null</li> <li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li> <li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Description</b>   | <p>TipcSrvMonServerNamesSetWatch sets whether this RTclient is watching the RTserver names that match the value of the Monitor_Scope option. TipcSrvMonServerNamesSetWatch saves the watch status and sends a MON_SERVER_NAMES_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_SERVER_NAMES_STATUS message each time an RTserver is created or destroyed. An RTserver is considered created when it starts up (and usually connects to other RTservers). An RTserver is considered destroyed when it terminates or disconnects from other RTservers. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.</p> <p>Each MON_SERVER_NAMES_STATUS message contains four fields:</p> <ul style="list-style-type: none"> <li>• a STR_ARRAY field containing the current RTserver names</li> <li>• a STR_ARRAY field containing RTserver info strings of the form "<i>ident: user@node</i>" (for example, "RTserver: ssuser@workstation1")</li> <li>• a STR field containing the name of the created RTserver (or "" if not applicable)</li> <li>• a STR field containing the name of the destroyed RTserver (or "" if not applicable)</li> </ul> |

In addition to sending a `MON_SERVER_NAMES_STATUS` message each time an `RTserver` is created or destroyed, `RTserver` also sends an initial `MON_SERVER_NAMES_STATUS` message when the watch status is set to `TRUE`, so that the `RTclient` does not have to wait for the next `RTserver` name change to receive the first `MON_SERVER_NAMES_STATUS` message.

To determine if this `RTclient` is watching `RTserver` names, use `TipcSrvMonServerNamesGetWatch`. To poll once for `RTserver` names, use `TipcSrvMonServerNamesPoll`.

**Caution** `TipcSrvMonServerNamesSetWatch` sends a `MON_SERVER_NAMES_SET_WATCH` message to `RTserver`, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

**See Also** `TipcSrvMonServerNamesGetWatch`, `TipcSrvMonServerNamesPoll`

**Examples** This example uses `TipcSrvMonServerNamesSetWatch` to start watching `RTserver` names and creates a callback to process the incoming `MON_SERVER_NAMES_STATUS` messages:

```
/* ===== */
/*..process_mon_server_names_status -- process a MON_SERVER_NAMES_STATUS message */
void T_ENTRY process_mon_server_names_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR *server_names;
 T_INT4 num_server_names;
 T_STR *info_strs;
 T_STR created_server;
 T_STR destroyed_server;
 T_INT4 counter;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
```

```

/* both server_names and info_strs contains the same number of
elements */

if (!TipcMsgNextStrArray(data->msg, &server_names,
 &num_server_names)
 || !TipcMsgNextStrArray(data->msg, &info_strs,
 &num_server_names)
 || !TipcMsgNextStr(data->msg, &created_server)
 || !TipcMsgNextStr(data->msg, &destroyed_server)) {
 return; /* error */
}

TutOut("Got SERVER_NAMES status.\n");
for (counter = 0; counter < num_server_names; counter++) {
 TutOut("server_names[%d] = %s\n", counter,
server_names[counter]);
 TutOut("info_strs[%d] = %s\n", counter, info_strs[counter]);
}
TutOut("created RTserver = %s\n", created_server);
TutOut("destroyed RTserver = %s\n", destroyed_server);
} /* process_mon_server_names_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonServerNamesSetWatch(srv, TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_NAMES_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_server_names_status, NULL)
 == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```

# TipcSrvMonServerOptionPoll

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonServerOptionPoll — poll once for RTserver option information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Synopsis      | <pre>T_BOOL TipcSrvMonServerOptionPoll(srv, server_name, option_name) T_IPC_SRV srv; T_STR server_name; T_STR option_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to poll for option information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p> <p><i>option_name</i> — name of option to poll for information (use T_IPC_MON_ALL to indicate a poll of all options)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Return Values | TRUE if the poll for RTserver option information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Diagnostics   | <p>If TipcSrvMonServerOptionPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i>, <i>server_name</i>, or <i>option_name</i> was null</li><li>• any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description   | <p>TipcSrvMonServerOptionPoll polls for RTserver option information by sending a MON_SERVER_OPTION_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTserver responds by sending back a MON_SERVER_OPTION_POLL_RESULT message. The response should come back quickly. If <i>option_name</i> is T_IPC_MON_ALL then all options in the polled RTserver are polled. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_SERVER_OPTION_POLL_RESULT message contains one fixed field followed by one or more groups of five fields. The fixed field is a STR field containing the name of the RTserver. The group of five fields is:</p> <ul style="list-style-type: none"><li>• a STR field containing the name of the option</li><li>• an INT2 field containing the option type (really T_OPT_TYPE)</li></ul> |



- a STR field containing the value of the option, converted to a string like the setopt command displays
- an INT4 field containing whether the option is required to have a value (that is, cannot be UNKNOWN)
- a STR\_ARRAY field containing the legal values for the option; this array is empty if not applicable

There are multiple responses only if *server\_name* is a wildcard or T\_IPC\_MON\_ALL.

To poll once for RTclient option information, use TipcSrvMonClientOptionPoll.

For an example of how MON\_SERVER\_OPTION\_POLL\_RESULT messages can be used, see the RTmon GDI Server Information window.

**Caution** TipcSrvMonServerOptionPoll sends a MON\_SERVER\_OPTION\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

There is no response message if the RTserver named *server\_name* or the option named *option\_name* does not exist; the poll is silently dropped.

RTserver option information can only be polled, not watched.

**See Also** TipcSrvMonClientOptionPoll, TipcSrvConnMsgWrite

**Examples** This example uses TipcSrvMonServerOptionPoll to poll for all option information in all RTservers, creates a callback to process the incoming MON\_SERVER\_OPTION\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_option_poll_result -- process a
MON_SERVER_OPTION_POLL_RESULT message */
void T_ENTRY process_mon_server_option_poll_result(conn, data,
arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR server_name;
 T_STR option_name;
 T_INT2 type; /* really (T_OPT_TYPE) */
 T_STR value_str;
 T_INT4 required; /* really (T_BOOL) */
 T_STR *legal_vals;
 T_INT4 num_legal_vals; /* is zero if legal_vals not applicable */
 T_INT4 counter; /* loop counter */
```

```

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get first fixed field */
 if (!TipcMsgNextStr(data->msg, &server_name)) {
 return; /* error */
 }
 TutOut("Got SERVER_OPTION poll result.\n");
 TutOut("RTserver name = %s\n", server_name);

 /* loop through all groups */
 while (TipcMsgRead(data->msg,
 T_IPC_FT_STR, &option_name,
 T_IPC_FT_INT2, &type,
 T_IPC_FT_STR, &value_str,
 T_IPC_FT_INT4, &required,
 T_IPC_FT_STR_ARRAY, &legal_vals,
 &num_legal_vals,
 NULL)) {
 TutOut("option name = %s\n", option_name);
 TutOut("type = %d\n", type);
 TutOut("value str = %s\n", value_str);
 TutOut("required = %s\n", required ? "TRUE" : "FALSE");
 if (num_legal_vals > 0) {
 TutOut("legal vals:\n");
 for (counter = 0; counter < num_legal_vals; counter++) {
 TutOut(" %s\n", legal_vals[counter]);
 }
 }
 }

 /* make sure we reached the end of the message */
 if (TutErrNumGet() != T_ERR_MSG_EOM) {
 return; /* error */
 }
} /* process_mon_server_option_poll_result */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

```

```

/* send the poll request out to RTserver */
if (!TipcSrvMonServerOptionPoll(srv, "...", T_IPC_MON_ALL)) {
 return; /* error */
}
mt = TipcMtLookupByNum(T_MT_MON_SERVER_OPTION_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_server_option_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

# TipcSrvMonServerRoutePoll

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonServerRoutePoll — poll once for RTserver route information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Synopsis      | <pre>T_BOOL TipcSrvMonServerRoutePoll(srv, server_name, dest_server_name) T_IPC_SRV srv; T_STR server_name; T_STR dest_server_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to poll for route information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p> <p><i>dest_server_name</i> — name of destination RTserver whose route in RTserver to poll (use T_IPC_MON_ALL to indicate a poll of routes to all other RTservers)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Return Values | TRUE if the poll for RTserver route information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Diagnostics   | <p>If TipcSrvMonServerRoutePoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i>, <i>server_name</i>, or <i>dest_server_name</i> was null</li><li>• any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Description   | <p>TipcSrvMonServerRoutePoll polls for RTserver route information by sending a MON_SERVER_ROUTE_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The route information to the RTserver <i>dest_server_name</i> is polled. The polled RTserver responds by sending back a MON_SERVER_ROUTE_POLL_RESULT message. The response should come back quickly. If <i>dest_server_name</i> is T_IPC_MON_ALL then the routes in the polled RTserver to all other RTservers are polled. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_SERVER_ROUTE_POLL_RESULT message contains one fixed field followed by one or more groups of four fields. The fixed field is a STR field containing the name of the RTserver. The group of four fields is:</p> <ul style="list-style-type: none"><li>• a STR field containing the name of the destination RTserver</li><li>• a STR_ARRAY field containing the names of the RTservers which are directly connected to the destination RTserver</li></ul> |

- an INT4 field containing the distance (in connections or hops) from the polled RTserver to the destination RTserver
- a STR\_ARRAY field containing the names of the RTservers to follow for the shortest path from the polled RTserver to the destination RTserver

There are multiple responses only if *server\_name* is a wildcard or T\_IPC\_MON\_ALL.

**Caution** TipcSrvMonServerRoutePoll sends a MON\_SERVER\_ROUTE\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message routing.

There is no response message if the RTserver named *server\_name* or destination RTserver named *dest\_server\_name* does not exist; the poll is silently dropped.

RTserver route information can only be polled, not watched.

**See Also** TipcSrvMonServerConnPoll, TipcSrvConnMsgWrite

**Examples** This example uses TipcSrvMonServerRoutePoll to poll for all route information in all RTservers, creates a callback to process the incoming MON\_SERVER\_ROUTE\_POLL\_RESULT messages, and then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_route_poll_result -- process a MON_SERVER_ROUTE_POLL_RESULT
message */
void T_ENTRY process_mon_server_route_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR server_name;
 T_STR dest_server_name;
 T_STR *connected_server_names;
 T_INT4 num_connected_server_names;
 T_INT4 distance;
 T_STR *route_to_server;
 T_INT4 num_route_to_server;
 T_INT4 counter;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }
}
```

```

/* get first fixed field */
if (!TipcMsgNextStr(data->msg, &server_name)) {
return; /* error */
}
TutOut("Got SERVER_ROUTE poll result.\n");
TutOut("RTserver name = %s\n", server_name);

/* loop through all groups */
while (TipcMsgRead(data->msg,
 T_IPC_FT_STR, &dest_server_name,
 T_IPC_FT_STR_ARRAY, &connected_server_names,
 &num_connected_server_names,
 T_IPC_FT_INT4, &distance,
 T_IPC_FT_STR_ARRAY, &route_to_server,
 &num_route_to_server,
 NULL)) {
TutOut("destination server name = %s\n", dest_server_name);
TutOut("connected servers:\n");
for (counter = 0; counter < num_connected_server_names;
 counter++) {
TutOut(" %s\n", connect_server_names[counter]);
}
TutOut("distance = %d\n", distance);
TutOut("route:\n");
for (counter = 0; counter < num_route_to_server;
 counter++) {
TutOut(" %s\n", route_to_server[counter]);
}
}

/* make sure we reached the end of the message */
if (TutErrNumGet() != T_ERR_MSG_EOM) {
return; /* error */
}

} /* process_mon_server_route_poll_result */

```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonServerRoutePoll(srv, "...", T_IPC_MON_ALL)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_ROUTE_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_server_route_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

# TipcSrvMonServerStartTimePoll

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonServerStartTimePoll — poll once for RTserver start time and elapsed time                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Synopsis      | <pre>T_BOOL TipcSrvMonServerStartTimePoll(srv, server_name) T_IPC_SRV srv; T_STR server_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to poll for time information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Return Values | TRUE if the poll was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Diagnostics   | <p>If TipcSrvMonServerStartTimePoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> or <i>server_name</i> was null</li><li>• any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Description   | <p>TipcSrvMonServerStartTimePoll polls for RTserver start time and elapsed time by sending a MON_SERVER_START_TIME_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTserver responds by sending back a MON_SERVER_START_TIME_POLL_RESULT message. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_SERVER_START_TIME_POLL_CALL message contains three fields:</p> <ul style="list-style-type: none"><li>• a STR field containing the name of the RTserver</li><li>• a REAL8 field containing the start time of the RTserver</li><li>• a REAL8 field containing the elapsed time of the RTserver</li></ul> <p>There are multiple responses only if <i>server_name</i> is a wildcard or T_IPC_MON_ALL.</p> |



**Caution** TipcSrvMonServerStartTimePoll sends a MON\_SERVER\_START\_TIME\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

There is no response message if the RTserver named *server\_name* does not exist; the poll is silently dropped.

RTserver time information can be polled, but it cannot be watched.

**See Also** TipcMonServerStartTimePoll

**Examples** This example uses TipcSrvMonServerStartTimePoll to poll for start time and elapsed time in all RTservers, creates a callback to process the incoming MON\_SERVER\_START\_TIME\_POLL\_RESULT messages, then waits up to 10 seconds for the poll results:

```
void T_ENTRY ServerStartTimeCb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR server_name;
 T_REAL8 start_time;
 T_REAL8 elapsed;
 T_INT4 days = 0;
 T_INT4 hours = 0;
 T_INT4 minutes = 0;
 T_INT4 seconds = 0;

 TipcMsgNextStr(data->msg, &server_name);
 TipcMsgNextReal8(data->msg, &start_time);
 TipcMsgNextReal8(data->msg, &elapsed);

 if (elapsed > 86400.0) {
 days = (T_INT4)floor(elapsed / 86400);
 elapsed -= days*86400;
 }
 if (elapsed > 3600.0) {
 hours = (T_INT4)floor(elapsed / 3600);
 elapsed -= hours*3600;
 }
 if (elapsed > 60.0) {
 minutes = (T_INT4)floor(elapsed / 60);
 elapsed -= minutes*60;
 }
 seconds = elapsed;

 TutOut("RTserver was started %d days %d hours %d minutes %d
seconds ago\n",
 days, hours, minutes, seconds);
} /* ServerStartTimeCb */
```

```

/* ... code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_SERVER_START_TIME_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt, ServerStartTimeCb, NULL)
== NULL) {
 return; /* error */
}

TipcSrvMonServerStartTimePoll(srv, "/...");
TipcSrvConnFlush(srv);
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

## TipcSrvMonServerTimePoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonServerTimePoll — poll once for RTserver time information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonServerTimePoll(srv, server_name) T_IPC_SRV srv; T_STR server_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to poll for time information (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Return Values</b> | TRUE if the poll for RTserver time information was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Diagnostics</b>   | <p>If TipcSrvMonServerTimePoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>server_name</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>   | <p>TipcSrvMonServerTimePoll polls for RTserver time information by sending a MON_SERVER_TIME_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTserver responds by sending back a MON_SERVER_TIME_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>Each MON_SERVER_TIME_POLL_RESULT message contains four fields:</p> <ul style="list-style-type: none"> <li>• a STR field containing the name of the RTserver</li> <li>• a REAL8 field containing the wall time (the time from TutGetWallTime)</li> <li>• a STR field containing the wall time converted to a string using the full time format</li> <li>• a REAL8 field containing the amount of CPU time used</li> </ul> <p>There are multiple responses only if <i>server_name</i> is a wildcard or T_IPC_MON_ALL.</p> <p>To poll once for RTclient time information, use TipcSrvMonClientTimePoll.</p> |

**Caution** TipcSrvMonServerTimePoll sends a `MON_SERVER_TIME_POLL_CALL` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

There is no response message if the RTserver named *server\_name* does not exist; the poll is silently dropped.

RTserver time information can only be polled, not watched, as RTserver does not have the notion of current time as RTclients do.

**See Also** `TipcSrvMonClientTimePoll`, `TipcSrvConnMsgWrite`

**Examples** This example uses `TipcSrvMonServerTimePoll` to poll for time information in all RTservers, creates a callback to process the incoming `MON_SERVER_TIME_POLL_RESULT` messages, then waits up to 10 seconds for the poll results:

```
/* ===== */
/*..process_mon_server_time_poll_result -- process a MON_SERVER_TIME_POLL_RESULT
message */
void T_ENTRY process_mon_server_time_poll_result(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR server_name;
 T_REAL8 wall_time;
 T_STR wall_time_str;
 T_REAL8 cpu_time;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgRead(data->msg,
 T_IPC_FT_STR, &server_name,
 T_IPC_FT_REAL8, &wall_time,
 T_IPC_FT_STR, &wall_time_str,
 T_IPC_FT_REAL8, &cpu_time,
 NULL)) {
 return; /* error */
 }

 TutOut("Got SERVER_TIME poll result.\n");
 TutOut("RTserver name = %s\n", server_name);
 TutOut("wall time = %f\n", wall_time);
 TutOut("wall time str = %s\n", wall_time_str);
 TutOut("CPU time = %f\n", cpu_time);
} /* process_mon_server_time_poll_result */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

/* send the poll request out to RTserver */
if (!TipcSrvMonServerTimePoll(srv, "...")) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SERVER_TIME_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_server_time_poll_result,
 NULL) == NULL) {
 return; /* error */
}

/* read and process all poll results (as well as any other incoming
messages) that arrive within the next 10 seconds */
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}

```

# TipcSrvMonServerVersionPoll

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonServerVersionPoll — poll once for RTserver version information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Synopsis      | <pre>T_BOOL TipcSrvMonServerVersionPoll(srv, server_name) T_IPC_SRV srv; T_STR server_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>server_name</i> — name of RTserver to poll (wildcard names allowed, or use T_IPC_MON_ALL to indicate a poll of all RTservers matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Return Values | TRUE if the poll was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Diagnostics   | <p>If TipcSrvMonServerVersionPoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> or <i>server_name</i> was null</li><li>• any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Description   | <p>TipcSrvMonServerVersionPoll polls for RTserver version information by sending a MON_SERVER_VERSION_POLL_CALL message to the <i>server_name</i> RTserver. The <i>server_name</i> argument can be the unique subject name of a single RTserver, a wildcarded subject name to match many RTservers, or T_IPC_MON_ALL to poll all RTservers that match the value of the Monitor_Scope option. The polled RTserver responds by sending back a MON_SERVER_VERSION_POLL_RESULT message. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response. Each MON_SERVER_VERSION_POLL_RESULT message contains two fields:</p> <ul style="list-style-type: none"><li>• a STR field containing the name of the RTserver</li><li>• a INT4 field containing the version of the RTserver</li></ul> <p>There are multiple responses only if <i>server_name</i> is a wildcard or T_IPC_MON_ALL.</p> |
| Caution       | <p>TipcSrvMonServerVersionPoll sends a MON_SERVER_VERSION_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.</p> <p>There is no response message if the RTserver named <i>server_name</i> does not exist; the poll is silently dropped.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

RTserver version information can be polled, but it cannot be watched.

**See Also**    TipcMonClientVersionPoll, TipcMonServerVersionPoll

**Examples**    This example uses TipcSrvMonServerVersionPoll to poll for version information in all RTservers, creates a callback to process the incoming MON\_SERVER\_VERSION\_POLL\_RESULT messages, then waits up to 10 seconds for the poll results:

```
void T_ENTRY ServerVersionCb(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR server_name;
 T_INT4 version;

 TipcMsgNextStr(data->msg, &server_name);
 TipcMsgNextInt4(data->msg, &version);
 TutOut("%s: %d\n", server_name, version);
} /* ServerVersionCb */

/* ... code from calling function is below */

T_IPC_MT mt;

mt = TipcMtLookupByNum(T_MT_MON_SERVER_VERSION_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt, ServerVersionCb, NULL) ==
NULL) {
 return; /* error */
}

TipcSrvMonServerVersionPoll(srv, "/...");
TipcSrvConnFlush(srv);
if (!TipcSrvConnMainLoop(srv, 10.0)) {
 return; /* error */
}
```

# TipcSrvMonSetIdentStr

|               |                                                                                                                                                                                                                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonSetIdentStr — set the monitoring identification string of this process                                                                                                                                                                                                              |
| Synopsis      | <pre>T_BOOL TipcSrvMonSetIdentStr(<i>srv</i>, <i>ident_str</i>) T_IPC_SRV <i>srv</i>; T_STR <i>ident_str</i>;</pre>                                                                                                                                                                           |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>ident_str</i> — identification string</p>                                                                                                                                                                                             |
| Return Values | TRUE if the identification string was successfully set for this process, FALSE otherwise.                                                                                                                                                                                                     |
| Diagnostics   | <p>If TipcSrvMonSetIdentStr fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> or <i>ident_str</i> was null</li></ul>                                                                          |
| Description   | TipcSrvMonSetIdentStr sets the monitoring identification string for this process. This identification string is used as a descriptive name for the process when it is being monitored. See TipcSrvMonGetIdentStr for more information on monitoring identification strings.                   |
| Caution       | Each RTclient sets the monitoring identification string in TipcSrvMonSetIdentStr. This string is stored in RTserver and is only sent to RTserver when RTclient connects to RTserver. A process that calls TipcSrvMonSetIdentStr after calling TipcSrvConnOpen will be identified incorrectly. |
| See Also      | TipcSrvMonGetIdentStr                                                                                                                                                                                                                                                                         |
| Examples      | <p>This example sets the monitoring identification string for this process to "Acme Inc. Custom RTclient":</p> <pre>if (!TipcSrvMonSetIdentStr(<i>srv</i>, "Acme Inc. Custom RTclient")) {     return; /* error */ }</pre>                                                                    |



## TipcSrvMonSubjectNamesGetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonSubjectNamesGetWatch — determine if this RTclient is watching subject names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonSubjectNamesGetWatch(srv, watch_status_return) T_IPC_SRV srv; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Return Values</b> | TRUE if the subject names watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Diagnostics</b>   | <p>If TipcSrvMonSubjectNamesGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>watch_status_return</i> was null</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>   | <p>TipcSrvMonSubjectNamesGetWatch determines whether this RTclient is watching the subject names in the current project that match the value of the Monitor_Scope option. The watch status controls whether the RTclient receives a MON_SUBJECT_NAMES_STATUS message each time a subject is created or destroyed. A subject is considered created when the first RTclient starts subscribing to that subject. A subject is considered destroyed when the last RTclient stops subscribing to that subject.</p> <p>Each MON_SUBJECT_NAMES_STATUS message contains three fields:</p> <ul style="list-style-type: none"> <li>• a STR_ARRAY field containing the current subject names</li> <li>• a STR field containing the name of the created subject (" " if not applicable)</li> <li>• a STR field containing the name of the destroyed subject (" " if not applicable)</li> </ul> <p>Use TipcSrvMonSubjectNamesSetWatch to set whether this RTclient is watching subject names. To poll once for subject names, use TipcSrvMonSubjectNamesPoll.</p> |
| <b>Caution</b>       | If TipcSrvMonSubjectNamesGetWatch returns FALSE, it does not store a value in <i>watch_status_return</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>See Also</b>      | TipcSrvMonSubjectNamesPoll, TipcSrvMonSubjectNamesSetWatch                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

**Examples** This uses TipcSrvMonSubjectNamesGetWatch to get the status of watching subject names:

```
T_BOOL watch_status;
if (!TipcSrvMonSubjectNamesGetWatch(srv, &watch_status)) {
 return; /* error */
}
TutOut("This process is %s subject names.\n",
 watch_status ? "watching" : "not watching");
```

## TipcSrvMonSubjectNamesPoll

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonSubjectNamesPoll — poll once for subject names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonSubjectNamesPoll(srv) T_IPC_SRV srv;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Arguments</b>     | <i>srv</i> — connection handle to RTserver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Return Values</b> | TRUE if the poll for subject names was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Diagnostics</b>   | <p>If TipcSrvMonSubjectNamesPoll fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> was null</li> <li>• any error number from TipcSrvConnMsgWrite</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Description</b>   | <p>TipcSrvMonSubjectNamesPoll polls for the subject names in the current project that match the value of the Monitor_Scope option by sending a MON_SUBJECT_NAMES_POLL_CALL message to RTserver. RTserver responds by sending back a MON_SUBJECT_NAMES_POLL_RESULT message. The response should come back quickly. The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response. The MON_SUBJECT_NAMES_POLL_RESULT message contains one field, a STR_ARRAY field containing the current subject names.</p> <p>To watch for created and destroyed subject names, use TipcSrvMonSubjectNamesSetWatch.</p> |
| <b>Caution</b>       | TipcSrvMonSubjectNamesPoll sends a MON_SUBJECT_NAMES_POLL_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>See Also</b>      | TipcSrvMonSubjectNamesGetWatch, TipcSrvMonSubjectNamesSetWatch, TipcSrvConnMsgWrite                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

**Examples** This example uses TpcSrvMonSubjectNamesPoll to poll for subject names, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR *subject_names;
T_INT4 num_subject_names;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TpcSrvMonSubjectNamesPoll(srv)) {
 return; /* error */
}

mt = TpcMtLookupByNum(T_MT_MON_SUBJECT_NAMES_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TpcSrvConnMsgSearchType(srv, 10.0, mt);
if (msg == NULL) {
 return; /* error */
}

/* set the current field to the first field */
if (!TpcMsgSetCurrent(msg, 0)) {
 return; /* error */
}

if (!TpcMsgNextStrArray(msg, &subject_names, &num_subject_names))
{
 return; /* error */
}

TutOut("Got SUBJECT_NAMES poll result.\n");
for (counter = 0; counter < num_subject_names; counter++) {
 TutOut("subject_names[%d] = %s\n", counter,
 subject_names[counter]);
}

/* clean up */
if (!TpcMsgDestroy(msg)) {
 return; /* error */
}
```

## TipcSrvMonSubjectNamesSetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonSubjectNamesSetWatch — start or stop watching subject names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonSubjectNamesSetWatch(srv, watch_status) T_IPC_SRV srv; T_BOOL watch_status;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Return Values</b> | TRUE if the subject names watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Diagnostics</b>   | <p>If TipcSrvMonSubjectNamesSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> was null</li> <li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE</li> <li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>   | <p>TipcSrvMonSubjectNamesSetWatch sets whether this RTclient is watching the subject names in the current project that match the value of the Monitor_Scope option. TipcSrvMonSubjectNamesSetWatch saves the watch status and sends a MON_SUBJECT_NAMES_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_SUBJECT_NAMES_STATUS message each time a subject is created or destroyed. A subject is considered created when the first RTclient starts subscribing to that subject. A subject is considered destroyed when the last RTclient stops subscribing to that subject. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver.</p> <p>Each MON_SUBJECT_NAMES_STATUS message contains three fields:</p> <ul style="list-style-type: none"> <li>• a STR_ARRAY field containing the current subject names</li> <li>• a STR field containing the name of the created subject (or "" if not applicable)</li> <li>• a STR field containing the name of the destroyed subject (or "" if not applicable)</li> </ul> <p>In addition to sending a MON_SUBJECT_NAMES_STATUS message each time a subject is created or destroyed, RTserver also sends an initial MON_SUBJECT_NAMES_STATUS message when the watch status is set to TRUE, so that the RTclient does not have to wait for the next subject name change to receive the first MON_SUBJECT_NAMES_STATUS message.</p> |

To determine if this RTclient is watching subject names, use `TipcSrvMonSubjectNamesGetWatch`. To poll once for subject names, use `TipcSrvMonSubjectNamesPoll`.

**Caution** `TipcSrvMonSubjectNamesSetWatch` sends a `MON_SUBJECT_NAMES_SET_WATCH` message to RTserver, but does not explicitly flush the message. See `TipcSrvConnGetAutoFlushSize` for more information on message buffering.

**See Also** `TipcSrvMonSubjectNamesGetWatch`, `TipcSrvMonSubjectNamesPoll`

**Examples** This example uses `TipcSrvMonSubjectNamesSetWatch` to start watching subject names and creates a callback to process the incoming `MON_SUBJECT_NAMES_STATUS` messages:

```
/* ===== */
/*..process_mon_subject_names_status -- process a MON_SUBJECT_NAMES_STATUS message
*/
void T_ENTRY process_mon_subject_names_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR *subject_names;
 T_INT4 num_subject_names;
 T_STR created_subject;
 T_STR destroyed_subject;
 T_INT4 counter;

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgNextStrArray(data->msg, &subject_names,
 &num_subject_names)
 || !TipcMsgNextStr(data->msg, &created_subject)
 || !TipcMsgNextStr(data->msg, &destroyed_subject)) {
 return; /* error */
 }

 TutOut("Got SUBJECT_NAMES status.\n");
 for (counter = 0; counter < num_subject_names; counter++) {
 TutOut("subject_names[%d] = %s\n", counter,
 subject_names[counter]);
 }
 TutOut("created subject = %s\n", created_subject);
 TutOut("destroyed subject = %s\n", destroyed_subject);
} /* process_mon_subject_names_status */
```

```

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonSubjectNamesSetWatch(srv, TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SUBJECT_NAMES_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_subject_names_status,
 NULL)
 == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```

# TipcSrvMonSubjectSubscribeGetWatch

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonSubjectSubscribeGetWatch — determine if this RTclient is watching the RTclients that are subscribing to a subject                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Synopsis      | <pre>T_BOOL TipcSrvMonSubjectSubscribeGetWatch(srv, subject_name, watch_status_return) T_IPC_SRV srv; T_STR subject_name; T_BOOL *watch_status_return;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>subject_name</i> — name of subject to get watch status for (use T_IPC_MON_ALL to indicate a watch of all subjects matching the value of Monitor_Scope)</p> <p><i>watch_status_return</i> — storage for watch status</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Return Values | TRUE if the subject subscribe watch status was successfully retrieved, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Diagnostics   | <p>If TipcSrvMonSubjectSubscribeGetWatch fails, it returns FALSE and sets the global SmartSockets error number to:</p> <ul style="list-style-type: none"><li>T_ERR_NULL_PTR — <i>srv</i>, <i>subject_name</i>, or <i>watch_status_return</i> was null</li><li>T_ERR_VAL_INVALID — <i>subject_name</i> was not a valid subject</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description   | <p>TipcSrvMonSubjectSubscribeGetWatch determines whether this RTclient is watching the RTclients that are subscribing to a subject. The watch status controls whether the RTclient receives a MON_SUBJECT_SUBSCRIBE_STATUS message each time an RTclient starts or stops subscribing to the subject. If <i>subject_name</i> is T_IPC_MON_ALL then all subjects in the current project that match the value of the Monitor_Scope option are watched.</p> <p>Each MON_SUBJECT_SUBSCRIBE_STATUS message contains four fields:</p> <ul style="list-style-type: none"><li>a STR field containing the name of the subject</li><li>a STR_ARRAY field containing the names of the current RTclients that are subscribing to the subject</li><li>a STR field containing the name of the RTclient that just started subscribing to the subject (or "" if not applicable)</li><li>a STR field containing the name of the RTclient that just stopped subscribing to the subject (or "" if not applicable)</li></ul> |



To set whether this RTclient is watching the RTclients that are subscribing to a subject, use `TipcSrvMonSubjectSubscribeSetWatch`. To poll once for the RTclients that are subscribing to a subject, use `TipcSrvMonSubjectSubscribePoll`.

For an example of how `MON_SUBJECT_SUBSCRIBE_STATUS` messages can be used, see the RTmon GDI Watch Clients in Subject window.

**Caution** If `TipcSrvMonSubjectSubscribeGetWatch` returns `FALSE`, it does not store a value in `watch_status_return`.

**See Also** `TipcSrvMonSubjectSubscribePoll`, `TipcSrvMonSubjectSubscribeSetWatch`

**Examples** This example uses `TipcSrvMonSubjectSubscribeGetWatch` to get the status of watching the RTclients that are subscribing to the thermal subject:

```
T_BOOL watch_status;

if (!TipcSrvMonSubjectSubscribeGetWatch(srv, "thermal",
&watch_status)) {
 return; /* error */
}
TutOut("This process is %s subject subscribers for thermal.\n",
 watch_status ? "watching" : "not watching");
```

# TipcSrvMonSubjectSubscribePoll

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcSrvMonSubjectSubscribePoll — poll once for the RTclients that are subscribing to a subject                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Synopsis      | <pre>T_BOOL TipcSrvMonSubjectSubscribePoll(srv, subject_name) T_IPC_SRV srv; T_STR subject_name;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Arguments     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>subject_name</i> — name of subject to poll for subscribers (use T_IPC_MON_ALL to indicate a poll of all subjects matching the value of Monitor_Scope)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Return Values | TRUE if the poll for the RTclients that are subscribing to a subject was successfully initiated, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Diagnostics   | <p>If TipcSrvMonSubjectSubscribePoll fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>srv</i> or <i>subject_name</i> was null</li><li>• any error number from TipcSrvConnMsgWrite</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Description   | <p>TipcSrvMonSubjectSubscribePoll polls for RTclients that are subscribing to a subject by sending a MON_SUBJECT_SUBSCRIBE_POLL_CALL message to RTserver. If <i>subject_name</i> is T_IPC_MON_ALL then all subjects in the current project that match the value of the Monitor_Scope option are polled. RTserver responds by sending back a MON_SUBJECT_SUBSCRIBE_POLL_RESULT message. The response should come back quickly.</p> <p>The polling RTclient can use functions such as TipcSrvConnMsgSearchType or TipcSrvConnMainLoop to get the response.</p> <p>The MON_SUBJECT_SUBSCRIBE_POLL_RESULT message contains groups of two fields:</p> <ul style="list-style-type: none"><li>• a STR field containing the name of the subject</li><li>• a STR_ARRAY field containing the names of the current RTclients that are subscribing to the subject</li></ul> <p>The group of two fields is repeated multiple times only if <i>subject_name</i> is T_IPC_MON_ALL.</p> <p>To watch for subject subscribers, use TipcSrvMonSubjectSubscribeSetWatch.</p> |

**Caution** TipcSrvMonSubjectSubscribePoll sends a MON\_SUBJECT\_SUBSCRIBE\_POLL\_CALL message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

**See Also** TipcSrvMonSubjectSubscribeGetWatch, TipcSrvMonSubjectSubscribeSetWatch, TipcSrvConnMsgWrite

**Examples** This example uses TipcSrvMonSubjectSubscribePoll to poll all subjects for the RTclients that are subscribing to the subject, waits up to 10 seconds for the poll result, and then prints the poll result message:

```
T_IPC_MT mt;
T_IPC_MSG msg;
T_STR subject_name;
T_STR *subscribe_client_names;
T_INT4 num_subscribe_client_names;
T_INT4 counter;

/* send the poll request out to RTserver */
if (!TipcSrvMonSubjectSubscribePoll(srv, "/...")) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SUBJECT_SUBSCRIBE_POLL_RESULT);
if (mt == NULL) {
 return; /* error */
}

/* wait up to 10 seconds for the poll result */
msg = TipcSrvConnMsgSearchType(srv, 10.0, mt);
if (msg == NULL) {
 return; /* error */
}

/* set the current field to the first field */
if (!TipcMsgSetCurrent(msg, 0)) {
 return; /* error */
}

/* loop through all groups */
TutOut("Got SUBJECT_SUBSCRIBE poll result.\n");
while (TipcMsgNextStr(msg, &subject_name)
 && TipcMsgNextStrArray(msg, &subscribe_client_names,
 &num_subscribe_client_names)) {
 TutOut("subject name = %s\n", subject_name);
 for (counter = 0; counter < num_subscribe_client_names;
 counter++) {
 TutOut("subscribe_client_names[%d] = %s\n", counter,
 subscribe_client_names[counter]);
 }
}
```

```
/* make sure we reached the end of the message */
if (TutErrNumGet() != T_ERR_MSG_EOM) {
 return; /* error */
}

/* clean up */
if (!TpcMsgDestroy(msg)) {
 return; /* error */
}
```

## TipcSrvMonSubjectSubscribeSetWatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSrvMonSubjectSubscribeSetWatch — start or stop watching the RTclients that are subscribing to a subject                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSrvMonSubjectSubscribeSetWatch(<i>srv</i>, <i>subject_name</i>, <i>watch_status</i>) T_IPC_SRV <i>srv</i>; T_STR <i>subject_name</i>; T_BOOL <i>watch_status</i>;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>     | <p><i>srv</i> — connection handle to RTserver</p> <p><i>subject_name</i> — name of subject to set watch status for (use T_IPC_MON_ALL to indicate a watch of all subjects matching the value of Monitor_Scope)</p> <p><i>watch_status</i> — new watch status (TRUE to start watching, FALSE to stop watching)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Return Values</b> | TRUE if the subject subscribe watch status was successfully set, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b>   | <p>If TipcSrvMonSubjectSubscribeSetWatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"> <li>• T_ERR_NULL_PTR — <i>srv</i> or <i>subject_name</i> was null</li> <li>• T_ERR_VAL_INVALID — <i>watch_status</i> was not TRUE or FALSE, or <i>subject_name</i> was not a valid subject</li> <li>• T_ERR_VAL_SAME — <i>watch_status</i> is the same as the current watch status</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>   | <p>TipcSrvMonSubjectSubscribeSetWatch sets whether this RTclient is watching the RTclients that are subscribing to a subject. TipcSrvMonSubjectSubscribeSetWatch saves the watch status and sends a MON_SUBJECT_SUBSCRIBE_SET_WATCH message to RTserver. The watch status controls whether the RTclient receives a MON_SUBJECT_SUBSCRIBE_STATUS message each time an RTclient starts or stops subscribing to the subject. The watch status is saved internally so the RTclient can automatically resume watching all appropriate categories if RTclient has to reconnect to RTserver. If <i>subject_name</i> is T_IPC_MON_ALL then all subjects in the current project that match the value of the Monitor_Scope option are watched.</p> <p>Each MON_SUBJECT_SUBSCRIBE_STATUS message contains four fields:</p> <ul style="list-style-type: none"> <li>• a STR field containing the name of the subject</li> <li>• a STR_ARRAY field containing the names of the current RTclients that are subscribing to the subject</li> </ul> |

- a STR field containing the name of the RTclient that just started subscribing to the subject (or "" if not applicable)
- a STR field containing the name of the RTclient that just stopped subscribing to the subject (or "" if not applicable)

In addition to sending a MON\_SUBJECT\_SUBSCRIBE\_STATUS message each time an RTclient starts or stops subscribing to the subject, RTserver also sends an initial MON\_SUBJECT\_SUBSCRIBE\_STATUS message (or many messages if T\_IPC\_MON\_ALL is used) when the watch status is set to TRUE, so that the RTclient does not have to wait for the next subject subscriber change to receive the first MON\_SUBJECT\_SUBSCRIBE\_STATUS message.

To determine if this RTclient is watching the RTclients that are subscribing to a subject, use TipcSrvMonSubjectSubscribeGetWatch. To poll once for the RTclients that are subscribing to a subject, use TipcSrvMonSubjectSubscribePoll.

For an example of how MON\_SUBJECT\_SUBSCRIBE\_STATUS messages can be used, see the RTmon GDI Watch Clients in Subject window.

**Caution** TipcSrvMonSubjectSubscribeSetWatch sends a MON\_SUBJECT\_SUBSCRIBE\_SET\_WATCH message to RTserver, but does not explicitly flush the message. See TipcSrvConnGetAutoFlushSize for more information on message buffering.

There is no initial status message if the subject named *subject\_name* does not exist.

**See Also** TipcSrvMonSubjectSubscribeGetWatch, TipcSrvMonSubjectSubscribePoll

**Examples** This example uses TipcSrvMonSubjectSubscribeSetWatch to start watching the RTclients that are subscribing to the thermal subject and creates a callback to process the incoming MON\_SUBJECT\_SUBSCRIBE\_STATUS messages:

```
/* ===== */
/*..process_mon_subject_subscribe_status -- process a MON_SUBJECT_SUBSCRIBE_STATUS
message */
void T_ENTRY process_mon_subject_subscribe_status(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
{
 T_STR subject_name;
 T_STR *subscribe_client_names;
 T_INT4 num_subscribe_client_names;
 T_STR start_client;
 T_STR stop_client;
 T_INT4 counter;
```

```

 /* must set current field first */
 if (!TipcMsgSetCurrent(data->msg, 0)) {
 return; /* error */
 }

 /* get the fields from the message */
 if (!TipcMsgNextStr(data->msg, &subject_name)
 || !TipcMsgNextStrArray(data->msg, &subscribe_client_names,
 &num_subscribe_client_names)
 || !TipcMsgNextStr(data->msg, &start_client)
 || !TipcMsgNextStr(data->msg, &stop_client)) {
 return; /* error */
 }

 TutOut("Got SUBJECT_SUBSCRIBE status.\n");
 TutOut("subject name = %s\n", subject_name);
 for (counter = 0; counter < num_subscribe_client_names;
 counter++) {
 TutOut("subscribe_client_names[%d] = %s\n", counter,
 subscribe_client_names[counter]);
 }
 TutOut("start RTclient = %s\n", start_client);
 TutOut("stop RTclient = %s\n", stop_client);
} /* process_mon_subject_subscribe_status */

/* ===== */
/*...code from calling function is below */

T_IPC_MT mt;

if (!TipcSrvMonSubjectSubscribeSetWatch(srv, "thermal", TRUE)) {
 return; /* error */
}

mt = TipcMtLookupByNum(T_MT_MON_SUBJECT_SUBSCRIBE_STATUS);
if (mt == NULL) {
 return; /* error */
}

if (TipcSrvConnProcessCbCreate(srv, mt,
 process_mon_subject_subscribe_status,
 NULL) == NULL) {
 return; /* error */
}

/* At this point TipcSrvConnMainLoop can be used to read and process messages. */

```





## Chapter 16 **TipcStrToDeliveryMode - TipcSubjectValid**

This chapter covers the TipcStr API functions:

- *TipcStrToDeliveryMode*
- *TipcStrToFt*
- *TipcStrToLbMode*
- *TipcSubjectGetUnique*
- *TipcSubjectMatch*
- *TipcSubjectValid*

# TipcStrToDeliveryMode

---

|               |                                                                                                                                                                                                                                                                                                                             |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcStrToDeliveryMode — convert a string to a message delivery mode                                                                                                                                                                                                                                                         |
| Synopsis      | <pre>T_BOOL TipcStrToDeliveryMode(<i>delivery_mode_str</i>, <i>delivery_mode_return</i>) T_STR <i>delivery_mode_str</i>; T_IPC_DELIVERY_MODE *<i>delivery_mode_return</i>;</pre>                                                                                                                                            |
| Arguments     | <p><i>delivery_mode_str</i> — delivery mode string to convert</p> <p><i>delivery_mode_return</i> — storage for delivery mode</p>                                                                                                                                                                                            |
| Return Values | TRUE if the string was successfully converted to a delivery mode, FALSE otherwise.                                                                                                                                                                                                                                          |
| Diagnostics   | <p>If TipcStrToDeliveryMode fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>delivery_mode_str</i> or <i>delivery_mode_return</i> was null</li><li>• T_ERR_VAL_INVALID — the string was not a valid delivery mode</li></ul> |
| Description   | TipcStrToDeliveryMode converts a string (such as <code>best_effort</code> ) to a message delivery mode (such as <code>T_IPC_DELIVERY_BEST_EFFORT</code> ). The string can be in any case. For example, <code>all</code> , <code>ALL</code> , and <code>All</code> are all converted to <code>T_IPC_DELIVERY_ALL</code> .    |
| Caution       | If TipcStrToDeliveryMode returns FALSE, it does not store a value in <i>delivery_mode_return</i> .                                                                                                                                                                                                                          |
| See Also      | TipcDeliveryModeToStr                                                                                                                                                                                                                                                                                                       |

**Examples** This example reads a string from standard input, converts it to a delivery mode, then sets the delivery mode of a message:

```
void read_delivery_mode(msg)
T_IPC_MSG msg;
{
 T_CHAR str[80];
 T_IPC_DELIVERY_MODE delivery_mode;

 printf("Enter delivery mode: ");
 fflush(stdout);

 if (fgets(str, sizeof(str), stdin) == NULL) {
 return; /* error */
 }
 if (strlen(str) > 0) {
 str[strlen(str) - 1] = '\0'; /* strip off newline */
 }

 if (!TipcStrToDeliveryMode(str, &delivery_mode)) {
 return; /* error */
 }

 if (!TipcMsgSetDeliveryMode(msg, delivery_mode)) {
 return; /* error */
 }
} /* read_delivery_mode */
```

# TipcStrToFt

---

|               |                                                                                                                                                                                                                                                                                           |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name          | TipcStrToFt — convert a string to a message field type                                                                                                                                                                                                                                    |
| Synopsis      | <pre>T_BOOL TipcStrToFt(str, ft_return) T_STR str; T_IPC_FT *ft_return;</pre>                                                                                                                                                                                                             |
| Arguments     | <p><i>str</i> — field type string to convert</p> <p><i>ft_return</i> — storage for field type</p>                                                                                                                                                                                         |
| Return Values | TRUE if the string was successfully converted to a field type, FALSE otherwise.                                                                                                                                                                                                           |
| Diagnostics   | <p>If TipcStrToFt fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>T_ERR_NULL_PTR — <i>str</i> was null or <i>ft_return</i> was null</li><li>T_ERR_FT_INVALID — the string was not a valid field type</li></ul> |
| Description   | TipcStrToFt converts a string (such as REAL8) to a message field type (such as T_IPC_FT_REAL8). The string can be in any case. (For example, str_array, STR_ARRAY, and Str_Array are all get converted to T_IPC_FT_STR_ARRAY.)                                                            |
| Caution       | If TipcStrToFt returns FALSE, it does not store a value in <i>ft_return</i> .                                                                                                                                                                                                             |
| See Also      | TipcFtToStr                                                                                                                                                                                                                                                                               |

**Examples** This example reads a string from a file, converts it to a field type, reads a field value from the file, then appends the field to a message:

```
void read_field(file, msg)
FILE *file;
T_IPC_MSG msg;
{
 T_CHAR str[80];
 T_IPC_FT type;
 T_REAL8 real8;

 if (fgets(str, sizeof(str), file) == NULL) {
 return; /* error */
 }
 if (strlen(str) > 0) {
 str[strlen(str) - 1] = '\0'; /* strip off newline */
 }

 if (!TipcStrToFt(str, &type)) {
 return; /* error */
 }

 switch (type) {
 /* read value from file and append to message */
 case T_IPC_FT_REAL8:
 if (fscanf(file, "%lf", &real8) != 1) {
 return; /* error */
 }
 if (!TipcMsgAppendReal8(msg, real8)) {
 return; /* error */
 }
 break;
 /* other field types go here */
 } /* switch */
} /* read_field */
```

# TipcStrToLbMode

|                      |                                                                                                                                                                                                                                                                                                                                          |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcStrToLbMode — convert a string to a message load balancing mode                                                                                                                                                                                                                                                                      |
| <b>Synopsis</b>      | <pre>T_BOOL TipcStrToLbMode(lb_mode_str, lb_mode_return) T_STR lb_mode_str; T_IPC_LB_MODE *lb_mode_return;</pre>                                                                                                                                                                                                                         |
| <b>Arguments</b>     | <p><i>lb_mode_str</i> — load balancing mode string to convert</p> <p><i>lb_mode_return</i> — storage for the load balancing mode</p>                                                                                                                                                                                                     |
| <b>Return Values</b> | TRUE if the string was successfully converted to a load balancing mode, FALSE otherwise.                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b>   | <p>If TipcStrToLbMode fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>lb_mode_str</i> was null or <i>lb_mode_return</i> was null</li><li>• T_ERR_VAL_INVALID — <i>lb_mode_str</i> did not contain a valid load balancing mode</li></ul> |
| <b>Description</b>   | TipcStrToLbMode converts a string (such as <code>weighted</code> ) to a message load balancing mode (such as <code>T_IPC_LB_WEIGHTED</code> ). The string can be in any case. (For example, <code>weighted</code> , <code>WEIGHTED</code> , and <code>Weighted</code> are all converted to <code>T_IPC_LB_WEIGHTED</code> .)             |
| <b>Caution</b>       | If TipcStrToLbMode returns FALSE, it does not store a value in <i>lb_mode_return</i> .                                                                                                                                                                                                                                                   |
| <b>See Also</b>      | TipcLbModeToStr                                                                                                                                                                                                                                                                                                                          |
| <b>Examples</b>      | <p>This example converts a string mode into a number load balancing mode:</p> <pre>if (!TipcStrToLbMode(str, &amp;lb_mode)) {     return; /* error */ }</pre>                                                                                                                                                                            |

## TipcSubjectGetUnique

---

|                      |                                                                                                                                                                                                             |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSubjectGetUnique — get the unique subject of the process                                                                                                                                                |
| <b>Synopsis</b>      | <code>T_STR TipcSubjectGetUnique()</code>                                                                                                                                                                   |
| <b>Arguments</b>     | None                                                                                                                                                                                                        |
| <b>Return Values</b> | The unique subject name of the process.                                                                                                                                                                     |
| <b>Diagnostics</b>   | None                                                                                                                                                                                                        |
| <b>Description</b>   | TipcSubjectGetUnique returns the absolute unique subject name of the process as defined by the Unique_Subject option.                                                                                       |
| <b>Caution</b>       | None                                                                                                                                                                                                        |
| <b>See Also</b>      | TipcSubjectValid, TipcSubjectMatch                                                                                                                                                                          |
| <b>Examples</b>      | <p>This example sets the unique subject, then prints it out:</p> <pre>TutCommandParseStr("setopt unique_subject myclient");<br/>TutOut("Unique Subject set to &lt;%s&gt;\n", TipcSubjectGetUnique());</pre> |

# TipcSubjectMatch

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSubjectMatch — determine whether two subject names are the same                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSubjectMatch(subject1, subject2, match_return) T_STR subject1; T_STR subject2; T_BOOL *match_return;</pre>                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>     | <p><i>subject1</i> — the first subject name to test</p> <p><i>subject2</i> — the second subject name to test</p> <p><i>match_return</i> — storage for the match status</p>                                                                                                                                                                                                                                                                                                                        |
| <b>Return Values</b> | TRUE if the subject names were compared successfully, FALSE otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Diagnostics</b>   | <p>If TipcSubjectMatch fails, it returns FALSE and sets the global SmartSockets error number to one of:</p> <ul style="list-style-type: none"><li>• T_ERR_NULL_PTR — <i>subject1</i>, <i>subject2</i> or <i>match_return</i> was null</li><li>• T_ERR_VAL_INVALID — the subject names were not valid subjects</li></ul>                                                                                                                                                                           |
| <b>Description</b>   | <p>TipcSubjectMatch compares two subject names for equality and stores the result in <i>match_return</i>. Subjects are case insensitive, so the subject names may be in any case. For example, /TIBCO and /tibco match.</p> <p>The subject names must be absolute (they must begin with a forward slash, '/') or TipcSubjectMatch returns FALSE and sets the global error code to T_ERR_VAL_INVALID.</p> <p>For more information on subjects, see the <i>TIBCO SmartSockets User's Guide</i>.</p> |
| <b>Caution</b>       | If TipcSubjectMatch returns FALSE, it does not store a value in <i>match_return</i> .                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>See Also</b>      | TipcSubjectValid, TipcSubjectGetUnique                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Examples</b>      | <p>This example compares two subject names for equivalency:</p> <pre>T_STR subject1 = "/stocks/t*"; T_STR subject2 = "/stocks/talr"; T_BOOL match;  /*  * This will result in a match.  */ if (!TipcSubjectMatch(subject1, subject2, &amp;match)) {     return; /* error */ }</pre>                                                                                                                                                                                                               |



## TipcSubjectValid

---

|                      |                                                                                                                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>          | TipcSubjectValid — check if a subject name is a valid subject                                                                                                                                                      |
| <b>Synopsis</b>      | <pre>T_BOOL TipcSubjectValid(subject) T_STR subject;</pre>                                                                                                                                                         |
| <b>Arguments</b>     | <i>subject</i> — the subject name to validate                                                                                                                                                                      |
| <b>Return Values</b> | TRUE if the subject name is a valid subject, FALSE otherwise.                                                                                                                                                      |
| <b>Diagnostics</b>   | None                                                                                                                                                                                                               |
| <b>Description</b>   | TipcSubjectValid checks that a subject name is a valid subject. Invalid subject names are either NULL or contain nothing but white space and subject delimiters (for example, '/').                                |
| <b>Caution</b>       | None                                                                                                                                                                                                               |
| <b>See Also</b>      | TipcSubjectMatch, TipcSubjectGetUnique                                                                                                                                                                             |
| <b>Examples</b>      | <p>This example checks if a subject name is valid:</p> <pre>T_STR subject = "/stocks/t*";  /*  * This will result in a valid subject name.  */ if (!TipcSubjectValid(subject)) {     /* invalid subject */ }</pre> |



# Index

## B

### basic security

- TipcSrvConnSetCredentials 1174
- TipcSrvConnSetUsernamePassword 1184
- TipcSrvSetUsernamePassword 1016

## C

### callback

- decode 16
- default 14
- description 14
- encode 16
- error 15
- process 14
- queue 15
- read 14
- server close 20
- server create 18
- server destroy 18, 20
- server names traverse 18, 1049, 1051
- server open 20
- subject 18, 19
- write 14

### case sensitivity xxxi

- on UNIX and Windows xxxi

### CC command 8

### CC environment variable 8

### changing

- Default\_Subject\_Prefix option 902, 913
- Unique\_Subject option 902, 913

### Client Information GDI window 217, 261, 1255, 1300

### Client Message Traffic GDI window 237, 1275

### Client Subject Message Traffic GDI window 265, 270, 1304, 1308

### Client\_Reconnect\_Timeout option 31

### compiling and linking 8

### compression

- setting for message types 870
- setting for messages 750

### connect IPC-specific command 188

### connections 13

- global 17
- logical names 49, 51
- multiple 20
- named 1072

### control message type processing 37

### convenience function

- TipcConnMsgWrite 11
- TipcConnMsgWriteVa 11
- TipcMsgRead 11
- TipcMsgReadVa 11
- TipcMsgWrite 11
- TipcMsgWriteVa 11
- TipcSrvMsgWrite 11
- TipcSrvMsgWriteVa 11

### conventions used in this manual xxix

### creation mode

- T\_IPC\_MSG\_FILE\_CREATE\_APPEND 13
- T\_IPC\_MSG\_FILE\_CREATE\_MODE 13
- T\_IPC\_MSG\_FILE\_CREATE\_READ 13
- T\_IPC\_MSG\_FILE\_CREATE\_WRITE 13
- T\_IPC\_MSG\_FILE\_CREATE\_WRITE\_BINARY 13

### customer support xxxii

## D

### data structure 6

- T\_IPC\_CONN 6
- T\_IPC\_CONN\_DECODE\_CB\_DATA 16
- T\_IPC\_CONN\_DECODE\_CB\_FUNC 16
- T\_IPC\_CONN\_DEFAULT\_CB\_DATA 14
- T\_IPC\_CONN\_DEFAULT\_CB\_FUNC 14

- T\_IPC\_CONN\_ENCODE\_CB\_DATA 16
- T\_IPC\_CONN\_ENCODE\_CB\_FUNC 16
- T\_IPC\_CONN\_ERROR\_CB\_DATA 15
- T\_IPC\_CONN\_ERROR\_CB\_FUNC 15
- T\_IPC\_CONN\_MSG\_CB\_DATA 14
- T\_IPC\_CONN\_MSG\_CB\_FUNC 14, 19
- T\_IPC\_CONN\_PROCESS\_CB\_DATA 14
- T\_IPC\_CONN\_PROCESS\_CB\_FUNC 14, 19
- T\_IPC\_CONN\_QUEUE\_CB\_DATA 15
- T\_IPC\_CONN\_QUEUE\_CB\_FUNC 15
- T\_IPC\_CONN\_READ\_CB\_DATA 14
- T\_IPC\_CONN\_READ\_CB\_FUNC 14
- T\_IPC\_CONN\_WRITE\_CB\_FUNC 14
- T\_IPC\_EVENT\_DATA 21
- T\_IPC\_EVENT\_FUNC 21
- T\_IPC\_MSG 6
- T\_IPC\_MSG\_FIELD 6
- T\_IPC\_MSG\_FILE 6
- T\_IPC\_MSG\_TRAV 10
- T\_IPC\_MT 6
- T\_IPC\_SRV\_CLOSE\_CB\_DATA 20
- T\_IPC\_SRV\_CLOSE\_CB\_FUNC 20
- T\_IPC\_SRV\_CONN\_STATUS 18
- T\_IPC\_SRV\_CREATE\_CB\_DATA 18
- T\_IPC\_SRV\_CREATE\_CB\_FUNC 18
- T\_IPC\_SRV\_DESTROY\_CB\_DATA 18
- T\_IPC\_SRV\_DESTROY\_CB\_FUNC 18
- T\_IPC\_SRV\_OPEN\_CB\_DATA 20
- T\_IPC\_SRV\_OPEN\_CB\_FUNC 20
- T\_IPC\_SRV\_SUBJECT\_CB\_DATA 19
- decode callback 16
- default callback 14
- Default\_Subject\_Prefix option 902, 913
- delivery modes
  - described 612
- disconnect IPC-specific command 188

## E

- Enable\_Control\_Msgs option 37
- encode callback 16
- enumerated type
  - T\_IPC\_DELIVERY\_ALL 12
  - T\_IPC\_DELIVERY\_BEST\_EFFORT 12
  - T\_IPC\_DELIVERY\_MODE 12
  - T\_IPC\_DELIVERY\_ORDERED 12
  - T\_IPC\_DELIVERY\_SOME 12
  - T\_IPC\_LB\_MODE 12
  - T\_IPC\_LB\_NONE 12
  - T\_IPC\_LB\_ROUND\_ROBIN 12
  - T\_IPC\_LB\_SORTED 12
  - T\_IPC\_LB\_WEIGHTED 12
  - T\_IPC\_SRV\_CONN\_FULL 17
  - T\_IPC\_SRV\_CONN\_NONE 17
  - T\_IPC\_SRV\_CONN\_STATUS 17
  - T\_IPC\_SRV\_CONN\_WARM 17
  - T\_IPC\_SRV\_LOG\_DATA 17
  - T\_IPC\_SRV\_LOG\_INTERNAL 17
  - T\_IPC\_SRV\_LOG\_STATUS 17
  - T\_IPC\_SRV\_LOG\_TYPE 17
  - T\_IPC\_SRV\_STOP\_ALL 17
  - T\_IPC\_SRV\_STOP\_CLIENTS 17
  - T\_IPC\_SRV\_STOP\_NORMAL 17
  - T\_IPC\_SRV\_STOP\_SERVERS 17
  - T\_IPC\_SRV\_STOP\_TYPE 17
  - T\_IPC\_TIMEOUT 13
  - T\_IPC\_TIMEOUT\_DELIVERY 13
  - T\_IPC\_TIMEOUT\_KEEP\_ALIVE 13
  - T\_IPC\_TIMEOUT\_READ 13
  - T\_IPC\_TIMEOUT\_WRITE 13
- error callback 15
- error handling 22
- error number 22
- events 21
- extension data, monitoring 213, 1251

## F

### field type

- T\_IPC\_FT 11
- T\_IPC\_FT\_BINARY 11
- T\_IPC\_FT\_CHAR 11
- T\_IPC\_FT\_INT2 11
- T\_IPC\_FT\_INT2\_ARRAY 11
- T\_IPC\_FT\_INT4 11
- T\_IPC\_FT\_INT4\_ARRAY 11
- T\_IPC\_FT\_INT8 11
- T\_IPC\_FT\_INT8\_ARRAY 11
- T\_IPC\_FT\_INTERNAL\_BOOL 11
- T\_IPC\_FT\_INTERNAL\_GROUP 11
- T\_IPC\_FT\_INTERNAL\_ID 11
- T\_IPC\_FT\_INTERNAL\_VERBOSE 11
- T\_IPC\_FT\_INVALID 11
- T\_IPC\_FT\_MSG 11
- T\_IPC\_FT\_MSG\_ARRAY 11
- T\_IPC\_FT\_REAL16 11
- T\_IPC\_FT\_REAL16\_ARRAY 11
- T\_IPC\_FT\_REAL4 11
- T\_IPC\_FT\_REAL4\_ARRAY 11
- T\_IPC\_FT\_REAL8 11
- T\_IPC\_FT\_REAL8\_ARRAY 11
- T\_IPC\_FT\_STR 11
- T\_IPC\_FT\_STR\_ARRAY 11
- T\_IPC\_PROP\_DELIVERY\_MODE 11
- T\_IPC\_PROP\_DELIVERY\_TIMEOUT 11
- T\_IPC\_PROP\_DEST 11
- T\_IPC\_PROP\_DEST\_SERVER 11
- T\_IPC\_PROP\_HEADER\_STR\_ENCODE 11
- T\_IPC\_PROP\_LB\_MODE 11
- T\_IPC\_PROP\_PRIORITY 11
- T\_IPC\_PROP\_SCOPE 11
- T\_IPC\_PROP\_SENDER 11
- T\_IPC\_PROP\_TYPE 11
- T\_IPC\_PROP\_USER\_PROP 11

### file names

- specifying xxxi

### function

- XtAppAddInput 79, 946, 1110
- XtAppMainLoop 79

### functions

- case-sensitivity xxxi

## G

### GDI window

- Client Information 217, 261, 1255, 1300
- Client Message Traffic 237, 1275
- Client Subject Message Traffic 265, 270, 1304, 1308
- Project Browser 317, 322, 1358, 1363
- Server Buffers 325, 1366
- Server Information 345, 370, 1388, 1413
- Watch Client Buffer 195, 200, 1230, 1235
- Watch Client Subjects 274, 282, 1312, 1320
- Watch Client Time 285, 290, 1324, 1329
- Watch Messages Received 226, 1264, 1266
- Watch Messages Sent 232, 234, 1270, 1272
- Watch Server Connections 335, 339, 355, 359, 1376, 1381, 1398, 1402
- Watch Subjects in Client 393, 1437, 1442

### global connections 17

- guaranteed message delivery files 186

## I

### identifiers

- case sensitivity xxxi

### interprocess communication 1

### IPC

- see interprocess communication

- Ipc\_Gmd\_Directory option 186

### IPC-specific command

- connect 188
- disconnect 188
- subscribe 188
- unsubscribe 188

## K

- keep alive 33

**L**

logical connection names 49, 51

**M**

make 9  
 message  
   fields  
     traversing 10  
   type  
     traversing 12  
 message file 13  
 message type 12  
   traversing 12  
 messages  
   case sensitivity xxxi  
 multiple connections 20

**N**

named connections 1072  
 naming conventions 3

**O**

options  
   case sensitivity xxxi  
   Client\_Reconnect\_Timeout 31  
   Default\_Subject\_Prefix 902, 913  
   Enable\_Control\_Msgs 37  
   Ipc\_Gmd\_Directory 186  
   Server\_Disconnect\_Mode 31  
   Server\_Start\_Max\_Tries 901  
   Time\_Format 284, 286, 289, 1323, 1325, 1329  
   Unique\_Subject 30, 902, 913  
   Zero\_Recv\_Gmd\_Failure 31

**P**

process callback 14  
 Project Browser GDI window 317, 322, 1358, 1363  
 properties table 21

**Q**

queue callback 15

**R**

read callback 14  
 rtlink shell script 8

**S**

search function  
   T\_IPC\_CONN\_MSG\_SEARCH\_FUNC 13  
 security  
   TipcSrvConnSetCredentials 1174  
   TipcSrvConnSetUsernamePassword 1184  
   TipcSrvSetUsernamePassword 1016  
 Server Buffers GDI window 325, 1366  
 server close callback 20  
 server create callback 18  
 server destroy callback 18, 20  
 Server Information GDI window 345, 370, 1388, 1413  
 server names traverse callback 18  
 server open callback 20  
 Server\_Disconnect\_Mode option 31  
 Server\_Names option  
   traverse callback create 1049  
   traverse callback lookup 1051  
 Server\_Start\_Max\_Tries option 901  
 shell commands  
   specifying xxxi  
 subject 18  
 subject callback 19

subject callbacks 18  
 subscribe IPC-specific command 188  
 support, contacting xxxii

## T

T\_IPC\_CONN data structure 6  
 T\_IPC\_CONN\_DECODE\_CB\_DATA data structure 16  
 T\_IPC\_CONN\_DECODE\_CB\_FUNC data structure 16  
 T\_IPC\_CONN\_DEFAULT\_CB\_DATA data structure 14  
 T\_IPC\_CONN\_DEFAULT\_CB\_FUNC data structure 14  
 T\_IPC\_CONN\_ENCODE\_CB\_DATA data structure 16  
 T\_IPC\_CONN\_ENCODE\_CB\_FUNC data structure 16  
 T\_IPC\_CONN\_ERROR\_CB\_DATA data structure 15  
 T\_IPC\_CONN\_ERROR\_CB\_FUNC data structure 15  
 T\_IPC\_CONN\_MSG\_CB\_DATA data structure 14  
 T\_IPC\_CONN\_MSG\_CB\_FUNC data structure 14, 19  
 T\_IPC\_CONN\_MSG\_SEARCH\_FUNC search function 13  
 T\_IPC\_CONN\_PROCESS\_CB\_DATA data structure 14  
 T\_IPC\_CONN\_PROCESS\_CB\_FUNC data structure 14, 19  
 T\_IPC\_CONN\_QUEUE\_CB\_DATA data structure 15  
 T\_IPC\_CONN\_QUEUE\_CB\_FUNC data structure 15  
 T\_IPC\_CONN\_READ\_CB\_DATA data structure 14  
 T\_IPC\_CONN\_READ\_CB\_FUNC data structure 14  
 T\_IPC\_CONN\_WRITE\_CB\_FUNC data structure 14  
 T\_IPC\_DELIVERY\_ALL enumerated type 12  
 T\_IPC\_DELIVERY\_BEST\_EFFORT enumerated type 12  
 T\_IPC\_DELIVERY\_MODE enumerated type 12  
 T\_IPC\_DELIVERY\_ORDERED enumerated type 12  
 T\_IPC\_DELIVERY\_SOME enumerated type 12  
 T\_IPC\_EVENT\_DATA data structure 21  
 T\_IPC\_EVENT\_FUNC data structure 21  
 T\_IPC\_FT field type 11  
 T\_IPC\_FT\_BINARY field type 11  
 T\_IPC\_FT\_CHAR field type 11  
 T\_IPC\_FT\_INT2 field type 11  
 T\_IPC\_FT\_INT2\_ARRAY field type 11  
 T\_IPC\_FT\_INT4 field type 11  
 T\_IPC\_FT\_INT4\_ARRAY field type 11  
 T\_IPC\_FT\_INT8 field type 11  
 T\_IPC\_FT\_INT8\_ARRAY field type 11  
 T\_IPC\_FT\_INTERNAL\_BOOL field type 11  
 T\_IPC\_FT\_INTERNAL\_GROUP field type 11  
 T\_IPC\_FT\_INTERNAL\_ID field type 11  
 T\_IPC\_FT\_INTERNAL\_VERBOSE field type 11  
 T\_IPC\_FT\_INVALID field type 11  
 T\_IPC\_FT\_MSG field type 11  
 T\_IPC\_FT\_MSG\_ARRAY field type 11  
 T\_IPC\_FT\_REAL16 field type 11  
 T\_IPC\_FT\_REAL16\_ARRAY field type 11  
 T\_IPC\_FT\_REAL4 field type 11  
 T\_IPC\_FT\_REAL4\_ARRAY field type 11  
 T\_IPC\_FT\_REAL8 field type 11  
 T\_IPC\_FT\_REAL8\_ARRAY field type 11  
 T\_IPC\_FT\_STR field type 11  
 T\_IPC\_FT\_STR\_ARRAY field type 11  
 T\_IPC\_LB\_MODE enumerated type 12  
 T\_IPC\_LB\_NONE enumerated type 12  
 T\_IPC\_LB\_ROUND\_ROBIN enumerated type 12  
 T\_IPC\_LB\_SORTED enumerated type 12  
 T\_IPC\_LB\_WEIGHTED enumerated type 12  
 T\_IPC\_MSG data structure 6  
 T\_IPC\_MSG\_FIELD data structure 6  
 T\_IPC\_MSG\_FILE data structure 6  
 T\_IPC\_MSG\_FILE\_CREATE\_APPEND creation mode 13  
 T\_IPC\_MSG\_FILE\_CREATE\_MODE creation mode 13  
 T\_IPC\_MSG\_FILE\_CREATE\_READ creation mode 13  
 T\_IPC\_MSG\_FILE\_CREATE\_WRITE creation mode 13  
 T\_IPC\_MSG\_FILE\_CREATE\_WRITE\_BINARY creation mode 13  
 T\_IPC\_MSG\_TRAV data structure 10  
 T\_IPC\_MSG\_TRAV\_FUNC traversal function 10  
 T\_IPC\_MT data structure 6  
 T\_IPC\_MT\_TRAV\_FUNC traversal function 12  
 T\_IPC\_PROP\_DELIVERY\_MODE field type 11

- T\_IPC\_PROP\_DELIVERY\_TIMEOUT field type 11
- T\_IPC\_PROP\_DEST field type 11
- T\_IPC\_PROP\_DEST\_SERVER field type 11
- T\_IPC\_PROP\_HEADER\_STR\_ENCODE field type 11
- T\_IPC\_PROP\_LB\_MODE field type 11
- T\_IPC\_PROP\_PRIORITY field type 11
- T\_IPC\_PROP\_SCOPE field type 11
- T\_IPC\_PROP\_SENDER field type 11
- T\_IPC\_PROP\_TYPE field type 11
- T\_IPC\_PROP\_USER\_PROP field type 11
- T\_IPC\_SRV\_CLOSE\_CB\_DATA data structure 20
- T\_IPC\_SRV\_CLOSE\_CB\_FUNC data structure 20
- T\_IPC\_SRV\_CONN\_FULL enumerated type 17
- T\_IPC\_SRV\_CONN\_NONE enumerated type 17
- T\_IPC\_SRV\_CONN\_STATUS data structure 18
- T\_IPC\_SRV\_CONN\_STATUS enumerated type 17
- T\_IPC\_SRV\_CONN\_WARM enumerated type 17
- T\_IPC\_SRV\_CREATE\_CB\_DATA data structure 18
- T\_IPC\_SRV\_CREATE\_CB\_FUNC data structure 18
- T\_IPC\_SRV\_DESTROY\_CB\_DATA data structure 18
- T\_IPC\_SRV\_DESTROY\_CB\_FUNC data structure 18
- T\_IPC\_SRV\_LOG\_DATA enumerated type 17
- T\_IPC\_SRV\_LOG\_INTERNAL enumerated type 17
- T\_IPC\_SRV\_LOG\_STATUS enumerated type 17
- T\_IPC\_SRV\_LOG\_TYPE enumerated type 17
- T\_IPC\_SRV\_OPEN\_CB\_DATA data structure 20
- T\_IPC\_SRV\_OPEN\_CB\_FUNC data structure 20
- T\_IPC\_SRV\_STOP\_ALL enumerated type 17
- T\_IPC\_SRV\_STOP\_CLIENTS enumerated type 17
- T\_IPC\_SRV\_STOP\_NORMAL enumerated type 17
- T\_IPC\_SRV\_STOP\_SERVERS enumerated type 17
- T\_IPC\_SRV\_STOP\_TYPE enumerated type 17
- T\_IPC\_SRV\_SUBJECT\_CB\_DATA data structure 19
- T\_IPC\_SRV\_SUBJECT\_TRAV\_FUNC traversal function 17
- T\_IPC\_TIMEOUT enumerated type 13
- T\_IPC\_TIMEOUT\_DELIVERY enumerated type 13
- T\_IPC\_TIMEOUT\_KEEP\_ALIVE enumerated type 13
- T\_IPC\_TIMEOUT\_READ enumerated type 13
- T\_IPC\_TIMEOUT\_WRITE enumerated type 13
- technical support xxxii
- Time\_Format option 284, 286, 289, 1323, 1325, 1329
- TipcBufMsgAppend function 24, 415, 635, 642
- TipcBufMsgNext function 25
- TipcCbConnProcessGmdFailure function 30
- TipcCbConnProcessKeepAlive function 33
- TipcCbSrvError function 35
- TipcCbSrvProcessControl function 37
- TipcCbSrvProcessGmdFailure function 39
- TipcConnAccept function 41
- TipcConnBufferGetReadSize function 42
- TipcConnBufferGetWriteSize function 43
- TipcConnCheck function 44
- TipcConnCreate function 47
- TipcConnCreateClient function 49
- TipcConnCreateServer function 51
- TipcConnDefaultCbCreate function 53
- TipcConnDefaultCbLookup function 55
- TipcConnDestroy function 56
- TipcConnErrorCbCreate function 58
- TipcConnErrorCbLookup function 60
- TipcConnFlush function 61
- TipcConnGetArch function 63
- TipcConnGetAutoFlushSize function 65
- TipcConnGetBlockMode function 66
- TipcConnGetGmdMaxSize function 68
- TipcConnGetGmdNumPending function 70
- TipcConnGetNode function 71
- TipcConnGetNumQueued function 72
- TipcConnGetPid function 73
- TipcConnGetSocket function 74
- TipcConnGetTimeout function 13, 75, 142
- TipcConnGetUniqueSubject function 77
- TipcConnGetUser function 78
- TipcConnGetXtSource function 79
- TipcConnGmdFileCreate function 81
- TipcConnGmdFileDelete function 85
- TipcConnGmdMsgDelete function 87
- TipcConnGmdMsgResend function 90
- TipcConnGmdResend function 93
- TipcConnKeepAlive function 95
- TipcConnLock function 97
- TipcConnMainLoop function 80, 98, 969
- TipcConnMsgInsert function 100
- TipcConnMsgNext function 80, 102
- TipcConnMsgProcess function 104
- TipcConnMsgSearch function 13, 106
- TipcConnMsgSearchType function 110
- TipcConnMsgSend function 12, 111
- TipcConnMsgSendRpc function 114



TpcConnMsgWrite convenience function 11  
 TpcConnMsgWrite function 118  
 TpcConnMsgWriteVa convenience function 11  
 TpcConnMsgWriteVa function 120, 989  
 TpcConnProcessCbCreate function 122  
 TpcConnProcessCbLookup function 124  
 TpcConnQueueCbCreate function 126  
 TpcConnQueueCbLookup function 128  
 TpcConnRead function 130  
 TpcConnReadCbCreate function 133  
 TpcConnReadCbLookup function 135  
 TpcConnSetAutoFlushSize function 137  
 TpcConnSetBlockMode function 138  
 TpcConnSetGmdMaxSize function 139  
 TpcConnSetSocket function 141  
 TpcConnSetTimeout function 13  
 TpcConnTrafficGetBytesRecv function 144  
 TpcConnTrafficGetBytesRecv8 function 144  
 TpcConnTrafficGetBytesSent function 145  
 TpcConnTrafficGetMsgsRecv function 146  
 TpcConnTrafficGetMsgsRecv8 function 146  
 TpcConnTrafficGetMsgsSent function 147  
 TpcConnUnlock function 148  
 TpcConnWriteCbCreate function 149  
 TpcConnWriteCbLookup function 151  
 TpcDeliveryModeToStr function 155  
 TpcDispatcherCreate function 156  
 TpcDispatcherCreateDetached function 157  
 TpcDispatcherDestroy function 158  
 TpcDispatcherDispatch function 159  
 TpcDispatcherMainLoop function 160  
 TpcDispatcherSrvAdd function 161  
 TpcDispatcherSrvRemove function 162  
 TpcEventCreate function 163  
 TpcEventCreateConn function 165  
 TpcEventCreateMsg function 167  
 TpcEventCreateMsgType function 169  
 TpcEventCreateSocket function 171  
 TpcEventCreateTimer function 173  
 TpcEventDestroy function 175  
 TpcEventGetCheckMode function 176  
 TpcEventGetConn function 177  
 TpcEventGetData function 178  
 TpcEventGetDispatcher function 179  
 TpcEventGetInterval function 180

TpcEventGetSocket function 181  
 TpcEventGetType function 182  
 TpcEventSetInterval function 183  
 TpcFtToStr function 184  
 TpcGetGmdDir function 186  
 TpcInitCommands function 188  
 TpcInitThreads function 189  
 TpcLbModeToStr function 190  
 TpcMonClientBufferGetWatch function 194  
 TpcMonClientBufferPoll function 196  
 TpcMonClientBufferSetWatch function 199  
 TpcMonClientCbPoll function 203  
 TpcMonClientCongestionGetWatch function 206  
 TpcMonClientCongestionSetWatch function 207  
 TpcMonClientCpuPoll function 210  
 TpcMonClientExtPoll function 213  
 TpcMonClientGeneralPoll function 216  
 TpcMonClientInfoPoll function 221  
 TpcMonClientMsgRecvGetWatch function 225  
 TpcMonClientMsgRecvSetWatch function 227  
 TpcMonClientMsgSendGetWatch function 231  
 TpcMonClientMsgSendSetWatch function 233  
 TpcMonClientMsgTrafficPoll function 236  
 TpcMonClientMsgTypeExPoll function 240  
 TpcMonClientMsgTypePoll function 245  
 TpcMonClientNamesGetWatch function 250  
 TpcMonClientNamesNumPoll function 252  
 TpcMonClientNamesPoll function 254  
 TpcMonClientNamesSetWatch function 256  
 TpcMonClientOptionPoll function 260  
 TpcMonClientSubjectExPoll function 264  
 TpcMonClientSubjectPoll function 269  
 TpcMonClientSubscribeGetWatch function 273  
 TpcMonClientSubscribeNumPoll function 275  
 TpcMonClientSubscribePoll function 278  
 TpcMonClientSubscribeSetWatch function 281  
 TpcMonClientTimeGetWatch function 284  
 TpcMonClientTimePoll function 286  
 TpcMonClientTimeSetWatch function 289  
 TpcMonClientVersionPoll function 292  
 TpcMonExtDelete function 294  
 TpcMonExtSetBinary function 295  
 TpcMonExtSetBool function 296  
 TpcMonExtSetBoolArray function 297  
 TpcMonExtSetInt2 function 298

- TipcMonExtSetInt2Array function 299
- TipcMonExtSetInt4 function 300
- TipcMonExtSetInt4Array function 301
- TipcMonExtSetInt8 function 302
- TipcMonExtSetInt8Array function 303
- TipcMonExtSetReal16 function 308
- TipcMonExtSetReal16Array function 309
- TipcMonExtSetReal4 function 304
- TipcMonExtSetReal4Array function 305
- TipcMonExtSetReal8 function 306
- TipcMonExtSetReal8Array function 307
- TipcMonExtSetStr function 310
- TipcMonExtSetStrArray function 311
- TipcMonExtSetUtf8 function 312
- TipcMonExtSetUtf8Array function 313
- TipcMonGetIdentStr function 314
- TipcMonPrintWatch function 316
- TipcMonProjectNamesGetWatch function 317
- TipcMonProjectNamesPoll function 319
- TipcMonProjectNamesSetWatch function 321
- TipcMonServerBufferGetWatch function 328
- TipcMonServerBufferPoll function 324
- TipcMonServerCongestionSetWatch function 330
- TipcMonServerConnGetWatch function 334
- TipcMonServerConnPoll function 336
- TipcMonServerConnSetWatch function 338
- TipcMonServerCpuPoll function 341
- TipcMonServerGeneralPoll function 344
- TipcMonServerMaxClientLicensesGetWatch function 349
- TipcMonServerMaxClientLicensesSetWatch function 351
- TipcMonServerNamesGetWatch function 362
- TipcMonServerNamesPoll function 364
- TipcMonServerNamesSetWatch function 366
- TipcMonServerOptionPoll function 369
- TipcMonServerRoutePoll function 373
- TipcMonServerStartTimePoll 376
- TipcMonServerTimePoll function 376, 379
- TipcMonServerMsgTrafficExPoll function 354
- TipcMonServerMsgTrafficPoll function 358
- TipcMonServerVersionPoll function 382
- TipcMonSetIdentStr function 384
- TipcMonSubjectNamesGetWatch function 385
- TipcMonSubjectNamesPoll function 387
- TipcMonSubjectNamesSetWatch function 389
- TipcMonSubjectSubscribeGetWatch function 392
- TipcMonSubjectSubscribePoll function 394
- TipcMonSubjectSubscribeSetWatch function 397
- TipcMsgAck function 403
- TipcMsgAddNamedBinary function 405, 570, 571, 572
- TipcMsgAddNamedBinaryPtr function 406
- TipcMsgAddNamedBool function 408
- TipcMsgAddNamedBoolArray function 409
- TipcMsgAddNamedBoolArrayPtr function 410
- TipcMsgAddNamedByte function 412
- TipcMsgAddNamedChar function 413
- TipcMsgAddNamedInt2 function 414
- TipcMsgAddNamedInt2ArrayPtr function 417
- TipcMsgAddNamedInt4 function 419
- TipcMsgAddNamedInt4Array function 420
- TipcMsgAddNamedInt4ArrayPtr function 422
- TipcMsgAddNamedInt8 function 424
- TipcMsgAddNamedInt8Array function 425, 427
- TipcMsgAddNamedMsg function 429
- TipcMsgAddNamedMsgArray function 430
- TipcMsgAddNamedMsgArrayPtr function 432
- TipcMsgAddNamedMsgPtr function 434
- TipcMsgAddNamedReal16 function 445
- TipcMsgAddNamedReal16Array function 446
- TipcMsgAddNamedReal16ArrayPtr function 448
- TipcMsgAddNamedReal4 function 435
- TipcMsgAddNamedReal4Array function 436
- TipcMsgAddNamedReal4ArrayPtr function 438
- TipcMsgAddNamedReal8 function 440
- TipcMsgAddNamedReal8Array function 441
- TipcMsgAddNamedReal8ArrayPtr function 443
- TipcMsgAddNamedRealTimestamp function 457
- TipcMsgAddNamedStr function 450
- TipcMsgAddNamedStrArray function 451
- TipcMsgAddNamedStrArrayPtr function 453
- TipcMsgAddNamedStrPtr function 455
- TipcMsgAddNamedTimestamp function 456
- TipcMsgAddNamedTimestampArrayPtr function 459
- TipcMsgAddNamedUtf8 function 462
- TipcMsgAddNamedUtf8ArrayPtr function 465
- TipcMsgAddNamedUtf8Ptr function 467
- TipcMsgAddNamedUtf8rArray function 463
- TipcMsgAddNamedXml function 468
- TipcMsgAddNamedXmlPtr function 469

TpcMsgAddUnknown function 461  
 TpcMsgAppendBinary function 473  
 TpcMsgAppendBinaryPtr function 475, 557  
 TpcMsgAppendBool function 477  
 TpcMsgAppendBoolArray function 479  
 TpcMsgAppendByte function 483  
 TpcMsgAppendChar function 485  
 TpcMsgAppendInt2 function 487  
 TpcMsgAppendInt2Array function 489  
 TpcMsgAppendInt2ArrayPtr function 491  
 TpcMsgAppendInt4 function 493, 499  
 TpcMsgAppendInt4Array function 495, 501  
 TpcMsgAppendInt4ArrayPtr function 497  
 TpcMsgAppendInt8ArrayPtr function 503  
 TpcMsgAppendMsg function 505  
 TpcMsgAppendMsgArray function 507  
 TpcMsgAppendMsgArrayPtr function 510  
 TpcMsgAppendMsgPtr function 513  
 TpcMsgAppendReal16ArrayPtr function 531  
 TpcMsgAppendReal4 function 515  
 TpcMsgAppendReal4Array function 517  
 TpcMsgAppendReal4ArrayPtr function 519  
 TpcMsgAppendReal8 function 521, 527  
 TpcMsgAppendReal8Array function 523, 529  
 TpcMsgAppendReal8ArrayPtr function 525  
 TpcMsgAppendStr function 533, 551  
 TpcMsgAppendStrArray function 535, 553  
 TpcMsgAppendStrArrayPtr function 481, 537, 555  
 TpcMsgAppendStrPtr function 539  
 TpcMsgAppendStrReal8 function 541, 549  
 TpcMsgAppendTimestamp function 543  
 TpcMsgAppendTimestampArray function 545  
 TpcMsgAppendTimestampArrayPtr function 547  
 TpcMsgAppendXml function 559  
 TpcMsgClone function 566  
 TpcMsgCreate function 568  
 TpcMsgDeleteCurrent function 570  
 TpcMsgDeleteField function 571  
 TpcMsgDeleteNamedField function 572  
 TpcMsgDestroy function 573  
 TpcMsgExistsNamed function 575  
 TpcMsgFieldSetSize function 576  
 TpcMsgFieldUpdateBinaryPtr function 577  
 TpcMsgFieldUpdateBoolArrayPtr function 578  
 TpcMsgFieldUpdateInt2ArrayPtr function 579

TpcMsgFieldUpdateInt4ArrayPtr function 580  
 TpcMsgFieldUpdateInt8ArrayPtr function 581  
 TpcMsgFieldUpdateMsgArrayPtr function 582  
 TpcMsgFieldUpdateMsgPtr function 583  
 TpcMsgFieldUpdateReal16ArrayPtr function 586  
 TpcMsgFieldUpdateReal4ArrayPtr function 584  
 TpcMsgFieldUpdateReal8ArrayPtr function 585  
 TpcMsgFieldUpdateStrArrayPtr function 588  
 TpcMsgFieldUpdateStrPtr function 589  
 TpcMsgFieldUpdateTimestampArrayPtr  
     function 590  
 TpcMsgFieldUpdateUtf8ArrayPtr function 591  
 TpcMsgFieldUpdateUtf8Ptr function 592  
 TpcMsgFieldUpdateXmlPtr function 593  
 TpcMsgFileCreate function 13, 594, 598  
 TpcMsgFileCreateFromFile function 13, 596  
 TpcMsgFileRead function 599  
 TpcMsgFileWrite function 601  
 TpcMsgGetCompression function 606  
 TpcMsgGetCurrent function 608  
 TpcMsgGetCurrentFieldValid function 610  
 TpcMsgGetDeliveryMode function 612  
 TpcMsgGetDeliveryTimeout function 614  
 TpcMsgGetDest function 615  
 TpcMsgGetHeaderStrEncode function 617  
 TpcMsgGetInt8Array function 637  
 TpcMsgGetLbMode function 618  
 TpcMsgGetNameCurrent function 621  
 TpcMsgGetNamedBinary function 622  
 TpcMsgGetNamedBool function 624  
 TpcMsgGetNamedBoolArray function 625  
 TpcMsgGetNamedByte function 627  
 TpcMsgGetNamedChar function 628  
 TpcMsgGetNamedInt2 function 629  
 TpcMsgGetNamedInt2Array function 630  
 TpcMsgGetNamedInt4 function 632  
 TpcMsgGetNamedInt4Array function 633  
 TpcMsgGetNamedMsg function 639  
 TpcMsgGetNamedMsgArray function 640  
 TpcMsgGetNamedReal16 function 648  
 TpcMsgGetNamedReal16Array function 650  
 TpcMsgGetNamedReal4Array function 643  
 TpcMsgGetNamedStr function 652  
 TpcMsgGetNamedStrArray function 653  
 TpcMsgGetNamedUnknown function 659

- TipcMsgGetNamedUtf8 function 660
- TipcMsgGetNamedUtf8Array function 661
- TipcMsgGetNamedXml function 663
- TipcMsgGetNumFields function 664
- TipcMsgGetPacketSize function 665
- TipcMsgGetPriority function 666
- TipcMsgGetReadOnly function 667
- TipcMsgGetRefCount function 668
- TipcMsgGetSender function 670
- TipcMsgGetSeqNum function 672
- TipcMsgGetType function 673
- TipcMsgGetTypeNamed function 675
- TipcMsgGetTypeNum function 676
- TipcMsgGetUserProp function 677
- TipcMsgIncrRefCount function 679
- TipcMsgNamedReal8 function 645
- TipcMsgNamedReal8Array function 646
- TipcMsgNamedTimestamp function 655
- TipcMsgNamedTimestampArray function 657
- TipcMsgNextBinary function 683
- TipcMsgNextChar function 691
- TipcMsgNextInt2 function 693
- TipcMsgNextInt2Array function 695
- TipcMsgNextInt4 function 697
- TipcMsgNextInt4Array function 699
- TipcMsgNextInt8 function 701
- TipcMsgNextInt8Array function 703
- TipcMsgNextMsg function 705
- TipcMsgNextMsgArray function 707
- TipcMsgNextReal16 function 717
- TipcMsgNextReal16Array function 719
- TipcMsgNextReal4 function 709
- TipcMsgNextReal4Array function 711
- TipcMsgNextReal8 function 713
- TipcMsgNextReal8Array function 715
- TipcMsgNextStr function 685, 689, 721, 735
- TipcMsgNextStrArray function 687, 723, 737
- TipcMsgNextStrReal8 function 725
- TipcMsgNextTimestamp function 727
- TipcMsgNextTimestampArray function 729
- TipcMsgNextType function 731
- TipcMsgNextUnknown function 733
- TipcMsgNextXml function 739
- TipcMsgPrint function 741
- TipcMsgPrintError function 743
- TipcMsgRead convenience function 11
- TipcMsgRead function 745
- TipcMsgReadVa convenience function 11
- TipcMsgReadVa function 747
- TipcMsgSetCompression function 750
- TipcMsgSetCurrent function 753
- TipcMsgSetDeliveryMode function 755
- TipcMsgSetDeliveryTimeout function 757
- TipcMsgSetDest function 758
- TipcMsgSetHeaderStrEncode function 760
- TipcMsgSetLbMode 761
- TipcMsgSetLbMode function 761
- TipcMsgSetNameCurrent function 763
- TipcMsgSetNumFields function 764
- TipcMsgSetPriority function 766
- TipcMsgSetSender function 768
- TipcMsgSetType function 770
- TipcMsgSetUserProp function 771
- TipcMsgTraverse function 10, 772
- TipcMsgUpdateNamedBinary function 777
- TipcMsgUpdateNamedBinaryPtr function 778
- TipcMsgUpdateNamedBool function 780
- TipcMsgUpdateNamedBoolArray function 781
- TipcMsgUpdateNamedBoolArrayPtr function 782
- TipcMsgUpdateNamedByte function 784
- TipcMsgUpdateNamedChar function 785
- TipcMsgUpdateNamedInt2 function 786
- TipcMsgUpdateNamedInt2Array function 787
- TipcMsgUpdateNamedInt2ArrayPtr function 788
- TipcMsgUpdateNamedInt4 function 790
- TipcMsgUpdateNamedInt4Array function 791
- TipcMsgUpdateNamedInt4ArrayPtr function 792
- TipcMsgUpdateNamedInt8 function 794
- TipcMsgUpdateNamedInt8Array function 795
- TipcMsgUpdateNamedInt8ArrayPtr function 796
- TipcMsgUpdateNamedMsg function 798
- TipcMsgUpdateNamedMsgArray function 799
- TipcMsgUpdateNamedMsgArrayPtr function 800
- TipcMsgUpdateNamedMsgPtr function 802
- TipcMsgUpdateNamedReal16 function 811
- TipcMsgUpdateNamedReal16Array function 812
- TipcMsgUpdateNamedReal16ArrayPtr function 814
- TipcMsgUpdateNamedReal4 function 803
- TipcMsgUpdateNamedReal4Array function 804
- TipcMsgUpdateNamedReal4ArrayPtr function 805

- TipcMsgUpdateNamedReal8 function 807
- TipcMsgUpdateNamedReal8Array function 808
- TipcMsgUpdateNamedReal8ArrayPtr function 809
- TipcMsgUpdateNamedStr function 816
- TipcMsgUpdateNamedStrArray function 817
- TipcMsgUpdateNamedStrArrayPtr function 818
- TipcMsgUpdateNamedStrPtr function 820
- TipcMsgUpdateNamedTimestamp function 821
- TipcMsgUpdateNamedTimestampArray function 822
- TipcMsgUpdateNamedTimestampArrayPtr function 823
- TipcMsgUpdateNamedUnknown function 825
- TipcMsgUpdateNamedUtf8 function 826
- TipcMsgUpdateNamedUtf8ArrayPtr function 828
- TipcMsgUpdateNamedUtf8Ptr function 827, 830
- TipcMsgUpdateNamedXml function 831
- TipcMsgUpdateNamedXmlPtr function 832
- TipcMsgWrite convenience function 11
- TipcMsgWrite function 833
- TipcMsgWriteVa convenience function 11
- TipcMsgWriteVa function 835
- TipcMtCreate function 841
- TipcMtDestroy function 844
- TipcMtGetCompression function 845
- TipcMtGetDeliveryMode function 847
- TipcMtGetDeliveryTimeout function 850
- TipcMtGetGrammar function 852
- TipcMtGetHeaderStrEncode function 853
- TipcMtGetLbMode function 855
- TipcMtGetName function 857
- TipcMtGetNum function 858
- TipcMtGetPriority function 859
- TipcMtGetUserProp function 861
- TipcMtLogAddMt function 863
- TipcMtLogRemoveMt function 865
- TipcMtLookup function 866
- TipcMtLookupByNum function 867
- TipcMtPrint function 868
- TipcMtSetCompression function 870
- TipcMtSetDeliveryMode function 872
- TipcMtSetDeliveryTimeout function 873
- TipcMtSetHeaderStrEncode function 874
- TipcMtSetLbMode function 875
- TipcMtSetPriority function 876
- TipcMtSetPriorityUnknown function 877
- TipcMtSetUserProp function 878
- TipcMtTraverse function 879
- TipcMtTraverse traversal function 12
- TipcPropertiesCreate function 882
- TipcPropertiesCreateMsg function 883
- TipcPropertiesDestroy function 884
- TipcPropertiesGet function 885
- TipcPropertiesGetCount function 886
- TipcPropertiesGetDefault function 887
- TipcPropertiesMsgCreate function 888
- TipcPropertiesSet function 889
- TipcPropertiesTraverse function 890
- TipcSrvBufferGetReadSize function 895
- TipcSrvBufferGetWriteSize function 896
- TipcSrvCheck function 897
- TipcSrvConnBufferGetReadSize function 1061
- TipcSrvConnBufferGetWriteSize function 1062
- TipcSrvConnCheck function 1063
- TipcSrvConnClose function 1065
- TipcSrvConnCloseCbCreate function 1068
- TipcSrvConnCloseCbLookup function 1070
- TipcSrvConnCreate function 1071
- TipcSrvConnCreateNamed function 1072
- TipcSrvConnDefaultCbCreate function 1073
- TipcSrvConnDefaultCbLookup function 1075
- TipcSrvConnDestroy function 1077
- TipcSrvConnErrorCbCreate function 1078
- TipcSrvConnErrorCbLookup function 1080
- TipcSrvConnFlush function 1082
- TipcSrvConnGetArch function 1083
- TipcSrvConnGetAutoFlushSize function 1085
- TipcSrvConnGetConnStatus function 1088
- TipcSrvConnGetCurrentConnectedLcn function 1087
- TipcSrvConnGetGmdDirName function 1089
- TipcSrvConnGetGmdMaxSize function 1090
- TipcSrvConnGetGmdNumPending function 1092
- TipcSrvConnGetNode function 1094
- TipcSrvConnGetNumQueued function 1096
- TipcSrvConnGetPid function 1098
- TipcSrvConnGetProject function 1100
- TipcSrvConnGetServerNamesList function 1101
- TipcSrvConnGetSocket function 1102
- TipcSrvConnGetTimeout function 1104
- TipcSrvConnGetUniqueSubject function 1106
- TipcSrvConnGetUser function 1108

TpcSrvConnGetXtSource function 1110  
 TpcSrvConnGmdFileCreate function 1112  
 TpcSrvConnGmdFileDelete function 1114  
 TpcSrvConnGmdMsgDelete function 1116  
 TpcSrvConnGmdMsgServerDelete function 1119  
 TpcSrvConnGmdMsgStatus function 1123  
 TpcSrvConnKeepAlive function 1128  
 TpcSrvConnLock function 1129  
 TpcSrvConnMainLoop function 1130  
 TpcSrvConnMsgInsert function 1132  
 TpcSrvConnMsgNext function 1134  
 TpcSrvConnMsgProcess function 1136  
 TpcSrvConnMsgSearch function 1138  
 TpcSrvConnMsgSearchType function 1140  
 TpcSrvConnMsgSend function 1142  
 TpcSrvConnMsgSendRpc function 1144  
 TpcSrvConnMsgWrite function 1148  
 TpcSrvConnMsgWriteVa function 1150  
 TpcSrvConnOpen function 1152  
 TpcSrvConnOpenCbCreate function 1154  
 TpcSrvConnOpenCbLookup function 1156  
 TpcSrvConnPrint function 1157  
 TpcSrvConnProcessCbCreate function 1159  
 TpcSrvConnProcessCbLookup function 1161  
 TpcSrvConnQueueCbCreate function 1163  
 TpcSrvConnQueueCbLookup function 1165  
 TpcSrvConnRead function 1167  
 TpcSrvConnReadCbCreate function 1168  
 TpcSrvConnReadCbLookup function 1170  
 TpcSrvConnSetAutoFlushSize function 1172  
 TpcSrvConnSetCredentials function 1174  
 TpcSrvConnSetGmdDirName function 1176  
 TpcSrvConnSetGmdMaxSize function 1177  
 TpcSrvConnSetProject function 1179  
 TpcSrvConnSetServerNames function 1180  
 TpcSrvConnSetServerNamesList function 1181  
 TpcSrvConnSetTimeout function 1182  
 TpcSrvConnSetUsernamePassword function 1184  
 TpcSrvConnStdSubjectSetSubscribe function 1185  
 TpcSrvConnSubjectCbCreate function 1187  
 TpcSrvConnSubjectCbDestroyAll function 1190  
 TpcSrvConnSubjectCbLookup function 1191  
 TpcSrvConnSubjectDefaultCbCreate function 1193  
 TpcSrvConnSubjectDefaultCbLookup function 1195  
 TpcSrvConnSubjectGetDefaultPrefix function 1196  
 TpcSrvConnSubjectGetSubscribe function 1197  
 TpcSrvConnSubjectGetSubscribeLb function 1199  
 TpcSrvConnSubjectGetUnique function 1201  
 TpcSrvConnSubjectGmdInit function 1202  
 TpcSrvConnSubjectLbInit function 1204  
 TpcSrvConnSubjectSetDefaultPrefix function 1206  
 TpcSrvConnSubjectSetSubscribe function 1207  
 TpcSrvConnSubjectSetSubscribeEx function 1209  
 TpcSrvConnSubjectSetSubscribeLb function 1211  
 TpcSrvConnSubjectSetUnique function 1213  
 TpcSrvConnSubjectTraverseSubscribe function 1214  
 TpcSrvConnTrafficGetBytesRecv function 1216  
 TpcSrvConnTrafficGetBytesRecv8 function 1216  
 TpcSrvConnTrafficGetBytesSent function 1217  
 TpcSrvConnTrafficGetBytesSent8 function 1217  
 TpcSrvConnTrafficGetMsgsRecv function 1218  
 TpcSrvConnTrafficGetMsgsRecv8 function 1218  
 TpcSrvConnTrafficGetMsgsSent function 1219  
 TpcSrvConnTrafficGetMsgsSent8 function 1219  
 TpcSrvConnUnlock function 1220  
 TpcSrvConnWriteCbCreate function 1221  
 TpcSrvConnWriteCbLookup function 1223  
 TpcSrvCreate function 17, 898  
 TpcSrvCreateCbCreate function 904  
 TpcSrvCreateCbLookup function 906  
 TpcSrvDefaultCbCreate function 907  
 TpcSrvDefaultCbLookup function 909  
 TpcSrvDestroy function 17, 911  
 TpcSrvDestroyCbCreate function 915  
 TpcSrvDestroyCbLookup function 918  
 TpcSrvErrorCbCreate function 919  
 TpcSrvErrorCbLookup function 921  
 TpcSrvFlush function 922  
 TpcSrvGetArch function 923  
 TpcSrvGetAutoFlushSize function 925  
 TpcSrvGetBlockMode function 927  
 TpcSrvGetConnStatus function 17, 929  
 TpcSrvGetCurrentConnectedLcn function 928  
 TpcSrvGetGmdMaxSize function 930  
 TpcSrvGetGmdNumPending function 932  
 TpcSrvGetNode function 934  
 TpcSrvGetNumQueued function 936  
 TpcSrvGetPid function 938  
 TpcSrvGetSocket function 939  
 TpcSrvGetTimeout function 940



- TipcSrvGetUniqueSubject function 942
- TipcSrvGetUser function 944
- TipcSrvGetXtSource function 946
- TipcSrvGmdFileCreate function 948
- TipcSrvGmdFileDelete function 950
- TipcSrvGmdMsgDelete function 952
- TipcSrvGmdMsgServerDelete function 955
- TipcSrvGmdMsgStatus function 958
- TipcSrvIsRunning function 963
- TipcSrvKeepAlive function 964
- TipcSrvLock function 965
- TipcSrvLogAddMt function 17, 966
- TipcSrvLogRemoveMt function 17, 968
- TipcSrvMonClientBufferGetWatch function 1229
- TipcSrvMonClientBufferPoll function 1231
- TipcSrvMonClientBufferSetWatch function 1234
- TipcSrvMonClientCbPoll function 1238
- TipcSrvMonClientCongestionGetWatch function 1242
- TipcSrvMonClientCongestionSetWatch function 1244
- TipcSrvMonClientCpuPoll function 1248
- TipcSrvMonClientExtPoll function 1251
- TipcSrvMonClientGeneralPoll function 1254
- TipcSrvMonClientInfoPoll function 1259
- TipcSrvMonClientMsgRecvGetWatch function 1263
- TipcSrvMonClientMsgRecvSetWatch function 1265
- TipcSrvMonClientMsgSendGetWatch function 1269
- TipcSrvMonClientMsgSendSetWatch function 1271
- TipcSrvMonClientMsgTrafficPoll function 1274
- TipcSrvMonClientMsgTypeExPoll function 1278
- TipcSrvMonClientMsgTypePoll function 1284
- TipcSrvMonClientNamesGetWatch function 1289
- TipcSrvMonClientNamesNumPoll function 1291
- TipcSrvMonClientNamesPoll function 1293
- TipcSrvMonClientNamesSetWatch function 1295
- TipcSrvMonClientOptionPoll function 1299
- TipcSrvMonClientSubjectExPoll function 1303
- TipcSrvMonClientSubjectPoll function 1307
- TipcSrvMonClientSubscribeGetWatch function 1311
- TipcSrvMonClientSubscribeNumPoll function 1313
- TipcSrvMonClientSubscribePoll function 1316
- TipcSrvMonClientSubscribeSetWatch function 1319
- TipcSrvMonClientTimeGetWatch function 1323
- TipcSrvMonClientTimePoll function 1325
- TipcSrvMonClientTimeSetWatch function 1328
- TipcSrvMonClientVersionPoll function 1332
- TipcSrvMonExtDelete function 1334
- TipcSrvMonExtSetBinary function 1335
- TipcSrvMonExtSetBool function 1336
- TipcSrvMonExtSetBoolArray function 1337
- TipcSrvMonExtSetInt2 function 1338
- TipcSrvMonExtSetInt2Array function 1339
- TipcSrvMonExtSetInt4 function 1340
- TipcSrvMonExtSetInt4Array function 1341
- TipcSrvMonExtSetInt8 function 1342
- TipcSrvMonExtSetInt8Array function 1343
- TipcSrvMonExtSetReal16 function 1348
- TipcSrvMonExtSetReal16Array function 1349
- TipcSrvMonExtSetReal4 function 1344
- TipcSrvMonExtSetReal4Array function 1345
- TipcSrvMonExtSetReal8 function 1346
- TipcSrvMonExtSetReal8Array function 1347
- TipcSrvMonExtSetStr function 1350
- TipcSrvMonExtSetStrArray function 1351
- TipcSrvMonExtSetUtf8 function 1352
- TipcSrvMonExtSetUtf8Array function 1353
- TipcSrvMonGetIdentStr function 1354
- TipcSrvMonPrintWatch function 1356
- TipcSrvMonProjectNamesGetWatch function 1358
- TipcSrvMonProjectNamesPoll function 1360
- TipcSrvMonProjectNamesSetWatch function 1362
- TipcSrvMonServerBufferPoll function 1365
- TipcSrvMonServerCongestionGetWatch  
function 1369
- TipcSrvMonServerCongestionSetWatch function 1371
- TipcSrvMonServerConnGetWatch function 1375
- TipcSrvMonServerConnPoll function 1377
- TipcSrvMonServerConnSetWatch function 1380
- TipcSrvMonServerCpuPoll function 1384
- TipcSrvMonServerGeneralPoll function 1387
- TipcSrvMonServerMaxClientLicensesGetWatch  
function 1392
- TipcSrvMonServerMaxClientLicensesSetWatch  
function 1394
- TipcSrvMonServerMsgTrafficExPoll function 1397
- TipcSrvMonServerMsgTrafficPoll function 1401
- TipcSrvMonServerNamesGetWatch function 1405
- TipcSrvMonServerNamesPoll function 1407
- TipcSrvMonServerNamesSetWatch function 1409
- TipcSrvMonServerOptionPoll function 1412
- TipcSrvMonServerRoutePoll function 1416

- TipcSrvMonServerStartTimePoll function 1420
- TipcSrvMonServerTimePoll function 1423
- TipcSrvMonServerMsgTrafficExPoll function 1397
- TipcSrvMonServerVersionPoll function 1426
- TipcSrvMonSetIdentStr function 1428
- TipcSrvMonSubjectNamesGetWatch function 1429
- TipcSrvMonSubjectNamesPoll function 1431
- TipcSrvMonSubjectNamesSetWatch function 1433
- TipcSrvMonSubjectSubscribeGetWatch function 1436
- TipcSrvMonSubjectSubscribePoll function 1438
- TipcSrvMonSubjectSubscribeSetWatch function 1441
- TipcSrvMsgInsert function 971
- TipcSrvMsgNext function 973
- TipcSrvMsgProcess function 975
- TipcSrvMsgSearch function 977
- TipcSrvMsgSearchType function 979
- TipcSrvMsgSend function 981
- TipcSrvMsgSendRpc function 983
- TipcSrvMsgWrite convenience function 11
- TipcSrvMsgWrite function 987
- TipcSrvMsgWriteVa convenience function 11
- TipcSrvPrint function 991
- TipcSrvProcessCbCreate function 993
- TipcSrvProcessCbLookup function 995
- TipcSrvQueueCbCreate function 997
- TipcSrvQueueCbLookup function 999
- TipcSrvRead function 1001
- TipcSrvReadCbCreate function 1002
- TipcSrvReadCbLookup function 1004
- TipcSrvSetAutoFlushSize function 1006
- TipcSrvSetBlockMode function 1008
- TipcSrvSetCredentials function 1009
- TipcSrvSetGmdMaxSize function 1011
- TipcSrvSetSocket function 1013
- TipcSrvSetTimeout function 1014
- TipcSrvSetUsernamePassword function 1016
- TipcSrvStdSubjectSetSubscribe function 1017
- TipcSrvStop function 17, 1019
- TipcSrvSubjectCbCreate function 1021
- TipcSrvSubjectCbDestroyAll function 1024
- TipcSrvSubjectCbLookup function 1025
- TipcSrvSubjectDefaultCbCreate function 1026
- TipcSrvSubjectDefaultCbLookup function 1028
- TipcSrvSubjectGetSubscribe function 1029
- TipcSrvSubjectGetSubscribeLb function 1031
- TipcSrvSubjectGmdInit function 1033
- TipcSrvSubjectLbInit function 1035
- TipcSrvSubjectSetSubscribe function 1037
- TipcSrvSubjectSetSubscribeEx function 1039
- TipcSrvSubjectSetSubscribeLb function 1041
- TipcSrvSubjectTraverseSubscribe function 17, 1043
- TipcSrvTrafficGetBytesRecv function 1045
- TipcSrvTrafficGetBytesRecv8 function 1045
- TipcSrvTrafficGetBytesSent function 1046
- TipcSrvTrafficGetBytesSent8 function 1046
- TipcSrvTrafficGetMsgsRecv function 1047
- TipcSrvTrafficGetMsgsRecv8 function 1047
- TipcSrvTrafficGetMsgsSent function 1048
- TipcSrvTrafficGetMsgsSent8 function 1048
- TipcSrvTraverseCbCreate function 1049
- TipcSrvTraverseCbLookup function 1051
- TipcSrvUnlock function 1052
- TipcSrvWriteCbCreate function 1053
- TipcSrvWriteCbLookup function 1055
- TipcStrToDeliveryMode function 1446
- TipcStrToFt function 1448
- TipcStrToLbMode function 1450
- TipcSubjectGetUnique function 1451
- TipcSubjectMatch function 1452
- TipcSubjectValid function 1453
- traversal function 10
  - T\_IPC\_MSG\_TRAV\_FUNC 10
  - T\_IPC\_MT\_TRAV\_FUNC 12
  - T\_IPC\_SRV\_SUBJECT\_TRAV\_FUNC 17
  - TipcMtTraverse 12
- TutErrNumGet function 22
- TutErrNumSet function 22
- TutErrNumToStr function 22
- TutErrStrGet function 22

## U

- Unique\_Subject option 30, 902, 913
- unsubscribe IPC-specific command 188



## V

varargs 747, 835

## W

Watch Client Buffer GDI window 195, 200, 1230, 1235

Watch Client Subjects GDI window 274, 282, 1312,  
1320

Watch Client Time GDI window 285, 290, 1324, 1329

Watch Messages Received GDI window 226, 1264,  
1266

Watch Messages Sent GDI window 232, 234, 1270,  
1272

Watch Server Connections GDI window 335, 339, 355,  
359, 1376, 1381, 1398, 1402

Watch Subjects in Client GDI window 393, 1437, 1442

write callback 14

## X

Xt Intrinsics

XtAppAddInput 79

XtAppMainLoop 79

XtAppAddInput function 946, 1110

## Z

Zero\_Recv\_Gmd\_Failure option 31

