

TIBCO SmartSockets™

ActiveX and COM Programmer's Guide

*Software Release 6.8
July 2006*

NOTE

All interfaces in this manual are deprecated. Though these interfaces will continue to be supported, they may be removed in a future release.

DO NOT use these interfaces for new development.

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.

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 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

Figures	xxi
Preface	xxiii
Related Documentation	xxiv
TIBCO Product Documentation	xxiv
Using the Online Documentation	xxiv
Conventions Used in This Manual	xxv
Typeface Conventions	xxv
Notational Conventions	xxvi
Identifiers	xxvi
Case	xxvii
How to Contact TIBCO Support	xxviii
Chapter 1 ActiveX and COM Environments	1
Overview	2
Loading the TIBCO SmartSockets ActiveX Components	3
Loading the TIBCO SmartSockets COM Library	5
TIBCO SmartSockets ActiveX Control Icons	6
Configuring Your Visual Basic Environment for ActiveX	7
Using Delphi Version 5 with ActiveX	8
Using PowerBuilder with ActiveX	10
Chapter 2 Using ActiveX Components	13
Getting Started	14
TIBCO SmartSockets ActiveX Classes	14
Writing a TIBCO SmartSockets ActiveX Program in Visual Basic	15
Testing the TIBCO SmartSockets ActiveX Program	20
Writing a TIBCO SmartSockets ActiveX Program in Delphi Version 5	21
Testing the TIBCO SmartSockets ActiveX Program	23
Writing a TIBCO SmartSockets ActiveX Program in PowerBuilder	23
Example Programs	27
SSChat: Multi-User Chat Room	28
WhoWhere: Electronic Message Board	33

Chapter 3 Using the COM Library..... 45

Getting Started 46

 TIBCO SmartSockets COM Classes 46

Writing a TIBCO SmartSockets COM Program in Visual Basic 47

Example Programs 54

 SSChat: Multi-User Chat Room..... 54

 WhoWhere: Electronic Message Board..... 59

Chapter 4 ActiveX Controls Reference..... 71

Relation to the TIBCO SmartSockets C API 72

TIBCO SmartSockets ActiveX Classes..... 72

TipcErrorCb Class 74

 TipcError Event 74

TipcMon Class 76

TipcMon Class Methods 77

 ClientBufferGetWatch Method 77

 ClientBufferPoll Method 78

 ClientBufferSetWatch Method 79

 ClientCbPoll Method 79

 ClientGeneralPoll Method 80

 ClientMsgRecvGetWatch Method 82

 ClientMsgRecvSetWatch Method 83

 ClientMsgSendGetWatch Method 83

 ClientMsgSendSetWatch Method 84

 ClientMsgTrafficPoll Method 85

 ClientMsgTypePoll Method 86

 ClientNamesGetWatch Method 87

 ClientNamesPoll Method 88

 ClientNamesSetWatch Method 89

 ClientOptionPoll Method 90

 ClientSubjectPoll Method..... 91

 ClientSubscribeGetWatch Method..... 92

 ClientSubscribePoll Method..... 93

 ClientSubscribeSetWatch Method..... 94

 ClientTimeGetWatch Method..... 94

 ClientTimePoll Method..... 95

 ClientTimeSetWatch Method..... 96

 GetIdentStr Method 97

 ProjectNamesGetWatch Method 97

 ProjectNamesPoll Method 98

 ProjectNamesSetWatch Method 99

 ServerBufferPoll Method 100

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

ServerConnGetWatch Method	101
ServerConnPoll Method	101
ServerConnSetWatch Method	102
ServerGeneralPoll Method	103
ServerMsgTrafficPoll Method	104
ServerNamesGetWatch Method	105
ServerNamesPoll Method	106
ServerNamesSetWatch Method	107
ServerOptionPoll Method	108
ServerRoutePoll Method	109
ServerTimePoll Method	110
SetIdentStr Method	111
SubjectNamesGetWatch Method	111
SubjectNamesPoll Method	112
SubjectNamesSetWatch Method	113
SubjectSubscribeGetWatch Method	113
SubjectSubscribePoll Method	114
SubjectSubscribeSetWatch Method	115
TipcMsg Class	117
TipcMsg Class Methods	119
Ack Method	119
AddNamedBinary Method	119
AddNamedChar Method	120
AddNamedInt2 Method	120
AddNamedInt2Array Method	121
AddNamedInt4 Method	122
AddNamedInt4Array Method	122
AddNamedMsg Method	123
AddNamedMsgArray Method	123
AddNamedReal4 Method	124
AddNamedReal4Array Method	125
AddNamedReal8 Method	125
AddNamedReal8Array Method	126
AddNamedStr Method	126
AddNamedStrArray Method	127
AddNamedXml Method	128
AppendBinary Method	128
AppendChar Method	129
AppendInt2 Method	129
AppendInt2Array Method	130
AppendInt4 Method	130
AppendInt4Array Method	131
AppendMsg Method	132
AppendMsgArray Method	132

AppendReal4 Method	133
AppendReal4Array Method	133
AppendReal8 Method	134
AppendReal8Array Method	135
AppendStr Method	135
AppendStrArray Method	136
AppendUnknown Method	136
Clone Method	137
Create Method	137
CurrentType Method	138
Destroy Method	138
GetNamedBinary Method	139
GetNamedChar Method.	139
GetNamedInt2 Method	140
GetNamedInt2Array Method	140
GetNamedInt4 Method	141
GetNamedInt4Array Method	142
GetNamedMsg Method	143
GetNamedMsgArray Method	143
GetNamedReal4 Method	144
GetNamedReal4Array Method.	145
GetNamedReal8 Method	145
GetNamedReal8Array Method.	146
GetNamedStr Method	147
GetNamedStrArray Method	147
GetNamedXml Method	148
IsNull Method	149
NextBinary Method	149
NextChar Method.	150
NextInt2 Method	150
NextInt2Array Method	151
NextInt4 Method	152
NextInt4Array Method	152
NextMsg Method	153
NextMsgArray Method	154
NextReal4 Method	155
NextReal4Array Method.	155
NextReal8 Method	156
NextReal8Array Method.	156
NextStr Method	157
NextStrArray Method	158
NextType Method.	159
NextUnknown Method	159

TipcMsg Class Properties	160
Current Property	160
CurrentUnknown Property	160
DeliveryMode Property	161
DeliveryTimeout Property	162
Dest Property	163
Fields Property	163
HeaderStrEncode Property	164
LbMode Property	165
NumFields Property	166
Priority Property	166
Sender Property	167
SeqNum Property	168
Type Property	168
UserProp Property	169
TipcMsgFields Class	170
TipcMsgFields Class Methods	171
Add Method	171
AddUnknown Method	172
TipcMsgFields Class Properties	173
Count Property	173
Item Property	173
TipcMt Class	175
TipcMt Class Methods	176
Create Method	176
Destroy Method	177
Lookup Method	177
LookupByNum Method	178
TipcMt Class Properties	179
DeliveryMode Property	179
DeliveryTimeout Property	180
Grammar Property	180
HeaderStrEncode Property	181
LbMode Property	182
Name Property	183
Num Property	183
Priority Property	184
UserProp Property	184
TipcProcessCb Class	186
TipcProcessCb Class Events	187
TipcMsg Event	187
TipcSrv Class	189

TipcSrv Class Methods 191

 Accept Method 191

 AddDefaultCb Method 191

 AddErrorCb Method 192

 AddProcessCb Method 193

 AddSubjectCb Method 194

 AddSubjectDefaultCb Method 195

 Check Method 196

 Create Method 196

 Destroy Method 197

 Flush Method 197

 GmdFileCreate Method 198

 GmdFileDelete Method 199

 GmdMsgDelete Method 199

 GmdMsgResend Method 200

 GmdMsgServerDelete Method 200

 GmdMsgStatus Method 201

 GmdResend Method 201

 KeepAlive Method 202

 LogAddMt Method 202

 LogRemoveMt Method. 203

 MainLoop Method 204

 MsgInsert Method 205

 MsgNext Method 206

 MsgProcess Method 206

 MsgSearchType Method 207

 MsgSend Method. 208

 MsgSendEx Method 208

 MsgSendRpc Method 209

 RemoveDefaultCb Method. 210

 RemoveErrorCb Method 211

 RemoveProcessCb Method. 211

 RemoveSubjectCb Method 212

 RemoveSubjectCbAll Method 213

 RemoveSubjectDefaultCb Method. 213

 StdSubjectRetrieve Method 214

 StdSubjectSubscribe Method. 214

 SubjectGmdInit Method 215

 SubjectLbInit Method 216

 SubjectRetrieve Method. 216

TipcSrv Class Properties	217
Arch Property	217
AutoFlushSize Property	217
BlockMode Property	218
ConnStatus Property	218
GmdDir Property	219
GmdMaxSize Property	220
GmdNumPending Property	220
Lcn Property	221
Node Property	221
NumQueued Property	222
Pid Property	222
Subjects Property	223
Timeout Property	223
UniqueSubject Property	224
User Property	225
TipcSubjects Class	226
TipcSubjects Class Methods	227
Add Method	227
Remove Method	228
TipcSubjects Class Properties	229
Count Property	229
Item Property	229
TipcSvc Class	231
TipcSvc Class Methods	232
AboutBox Method	232
CheckVersion Method	232
MsgCreate Method	233
MtCreate Method	233
MtLookup Method	234
ParseFile Method	235
ParseStdLicense Method	235
ParseStr Method	236
XmlCreate Method	236
TipcSvc Class Properties	238
AutoMainLoop Property	238
DefaultMsgPriority Property	239
DefaultSubjectPrefix Property	239
EnableControlMsgs Property	240
InputEventsStrategy Property	240
LastErrorNum Property	241
LastErrorStr Property	241
MonitorScope Property	242

Option Property	243
Project Property	243
ServerAutoConnect Property	244
ServerAutoFlushSize Property.	244
ServerDeliveryTimeout Property	245
ServerKeepAliveTimeout Property.	245
ServerMsgSend Property.	246
ServerNames Property	247
ServerReadTimeout Property	247
ServerStartDelay Property	248
ServerStartMaxTries Property	248
ServerStartTimeout Property	249
ServerWriteTimeout Property.	249
SocketConnectTimeout Property	250
Srv Property.	251
Subjects Property.	251
UdpBroadcastTimeout Property.	252
UniqueSubject Property.	252
TipcSvc Class Events	254
DataArrival Event.	254
SrvCreate Event	254
SrvDestroy Event.	256
TipcXml Class	257
TipcXml Class Methods	258
Create Method	258
Clone Method	258
Destroy Method	259
Chapter 5 COM Library Reference	261
Relation to the TIBCO SmartSockets C API	262
TIBCO SmartSockets COM Classes.	262
TipcConnChangeCb Class	264
TipcConnChangeCb Events	265
ConnChange Event	265
TipcDispatcher Class	267
TipcDispatcher Methods	268
Create Method	268
CreateDetached Method	268
Destroy Method	268
Dispatch Method	269
MainLoop Method	269

TipcErrorCb Class	271
TipcError Event	271
TipcEvent Class	273
TipcEvent Methods	274
Create Method	274
CreateMsg Method	275
CreateMsgType Method	276
CreateSocket Method	277
CreateTimer Method	278
Destroy Method	278
TipcEvent Properties	280
CheckMode Property	280
Data Property	280
Dispatcher Property	281
Interval Property	281
Socket Property	282
Type Property	282
TipcEventCb Class	283
TipcEventCb Events	284
DispatcherEvent Event	284
TipcEventMsgCb Class	286
TipcEventMsgCb Events	287
DispatcherMsgEvent Event	287
TipcMsg Class	289
TipcMsg Class Methods	292
Ack Method	292
AddNamedBinary Method	292
AddNamedChar Method	293
AddNamedInt2 Method	293
AddNamedInt2Array Method	294
AddNamedInt4 Method	295
AddNamedInt4Array Method	295
AddNamedInt8 Method	296
AddNamedInt8Array Method	296
AddNamedMsg Method	297
AddNamedMsgArray Method	298
AddNamedPersistedObject Method	298
AddNamedReal4 Method	299
AddNamedReal4Array Method	300
AddNamedReal8 Method	300
AddNamedReal8Array Method	301
AddNamedReal16 Method	301
AddNamedReal16Array Method	302

AddNamedStr Method	303
AddNamedStrArray Method	303
AddNamedTimestamp Method	304
AddNamedTimestampArray Method	304
AddNamedXml Method	305
AppendBinary Method	306
AppendChar Method	306
AppendInt2 Method	307
AppendInt2Array Method	307
AppendInt4 Method	308
AppendInt4Array Method	309
AppendInt8 Method	309
AppendInt8Array Method	310
AppendMsg Method	310
AppendMsgArray Method	311
AppendPersistedObject Method	312
AppendReal4 Method	312
AppendReal4Array Method	313
AppendReal8 Method	313
AppendReal8Array Method	314
AppendReal16 Method	314
AppendReal16Array Method	315
AppendStr Method	316
AppendStrArray Method	316
AppendTimestamp Method	317
AppendTimestampArray Method	317
AppendUnknown Method	318
AppendXML Method	319
Clone Method	319
CurrentType Method	320
CurrentUnknown Method	320
GetCompression Method	320
GetNamedBinary Method	321
GetNamedChar Method	322
GetNamedInt2 Method	322
GetNamedInt2Array Method	323
GetNamedInt4 Method	323
GetNamedInt4Array Method	324
GetNamedInt8 Method	325
GetNamedInt8Array Method	325
GetNamedMsg Method	326
GetNamedMsgArray Method	326
GetNamedPersistedObject Method	327
GetNamedReal4 Method	328
GetNamedReal4Array Method	328

GetNamedReal8 Method	329
GetNamedReal8Array Method	329
GetNamedReal16 Method	330
GetNamedReal16Array Method	331
GetNamedStr Method	331
GetNamedStrArray Method	332
GetNamedTimestamp Method	332
GetNamedTimestampArray Method	333
GetNamedXml Method	334
Load Method	334
NextBinary Method	336
NextChar Method	336
NextInt2 Method	337
NextInt2Array Method	337
NextInt4 Method	338
NextInt4Array Method	338
NextInt8 Method	339
NextInt8Array Method	339
NextMsg Method	340
NextMsgArray Method	341
NextPersistedObject Method	341
NextReal4 Method	342
NextReal4Array Method	342
NextReal8 Method	343
NextReal8Array Method	343
NextReal16 Method	344
NextReal16Array Method	345
NextStr Method	345
NextStrArray Method	346
NextTimestamp Method	346
NextTimestampArray Method	347
NextType Method	348
NextUnknown Method	348
NextXml Method	348
Reset Method	349
Save Method	349
SetCompression Method	350
UpdateNamedBinary Method	351
UpdateNamedCharacter Method	352
UpdateNamedInt2 Method	352
UpdateNamedInt2Array Method	353
UpdateNamedInt4 Method	353
UpdateNamedInt4Array Method	354
UpdateNamedInt8 Method	354
UpdateNamedInt8Array Method	355

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

- UpdateNamedMsg Method 356
- UpdateNamedMsgArray Method 356
- UpdateNamedPersistedObject Method 357
- UpdateNamedReal4 Method 358
- UpdateNamedReal4Array Method. 358
- UpdateNamedReal8 Method 359
- UpdateNamedReal8Array Method. 359
- UpdateNamedReal16 Method 360
- UpdateNamedReal16Array Method. 361
- UpdateNamedStr Method 361
- UpdateNamedStrArray Method 362
- UpdateNamedTimestamp Method 362
- UpdateNamedTimestampArray Method. 363
- UpdateNamedXML Method 364
- TipcMsg Class Properties 365
 - ArrivalTimestamp Property 365
 - CorrelationID Property 365
 - Current Property 366
 - CurrentUnknown Property 366
 - DeliveryMode Property 367
 - DeliveryTimeout Property 368
 - Dest Property 369
 - Fields Property. 369
 - HeaderStrEncode Property 370
 - LbMode Property 371
 - NumFields Property 372
 - Priority Property 373
 - ReplyTo Property 373
 - Sender Property. 374
 - SeqNum Property 374
 - Type Property 375
 - UserProp Property 376
- TipcMsgFields Class 377
- TipcMsgFields Class Methods 378
 - Add Method 378
 - AppendUnknown Method 379
- TipcMsgFields Class Properties 380
 - Count Property. 380
 - Item Property 380
- TipcMt Class 382

TipcMt Class Methods	383
Create Method	383
Destroy Method	383
GetCompression Method	384
Lookup Method	385
LookupByNum Method	386
SetCompression Method	386
TipcMt Class Properties	388
DeliveryMode Property	388
DeliveryTimeout Property	389
Grammar Property	390
HeaderStrEncode Property	390
LbMode Property	391
Name Property	393
Num Property	393
Priority Property	394
UserProp Property	394
TipcProcessCb Class	396
TipcProcessCb Class Events	397
TipcMsg Event	397
TipcSrvConn Class	399
TipcSrvConn Class Methods	401
CbDestroy Method	401
Check Method	401
Close Method	402
CloseCbCreate Method	403
Create Method	403
CreateNamed Method	404
DefaultCbCreate Method	404
Destroy Method	405
ErrorCbCreate Method	406
Flush Method	407
GetBufferReadSize Method	407
GetBufferWriteSize Method	408
GetSubscribe Method	409
GetSubscribeLb Method	409
GetTrafficBytesRecv Method	410
GetTrafficBytesSent Method	411
GetTrafficMsgsRecv Method	412
GetTrafficMsgsSent Method	413
GmdFileCreate Method	413
GmdFileDelete Method	414
GmdMsgDelete Method	415
GmdMsgServerDelete Method	415

GmdMsgStatus Method	416
KeepAlive Method	416
MainLoop Method	417
MsgInsert Method	418
MsgNext Method	418
MsgProcess Method	419
MsgSearchType Method	419
MsgSend Method.	420
MsgSendRpc Method	421
Open Method	422
OpenCbCreate Method	422
ProcessCbCreate Method	423
SetCredentials Method	424
SetStdSubjectSubscribe Method	425
SetUsernamePassword Method	425
SrvIsRunning Method	426
SrvLock Method.	426
TipcSrvMonCreate Method	427
TipcSrvMonExtCreate Method.	427
SrvUnlock Method	428
SubjectCbCreate Method.	428
SubjectCbDestroyAll Method.	429
SubjectDefaultCbCreate Method	430
SubjectGmdInit Method	431
SubjectLbInit Method	431
SubjectSetSubscribe Method.	432
SubjectSetSubscribeEx Method	433
SubjectSetSubscribeLb Method.	434
TipcSrvConn Class Properties	436
Arch Property	436
AutoFlushSize Property	436
BlockMode Property.	437
DefaultProtocols Property	437
DefaultSubjectPrefix Property	438
GmdMaxSize Property.	438
GmdNumPending Property	439
MonitorScope Property	439
Node Property	440
NumQueued Property	441
Pid Property.	441
Project Property	442
ServerAutoConnect Property.	442
ServerAutoFlushSize Property.	443
ServerDeliveryTimeout Property	443
ServerKeepAliveTimeout Property.	444

ServerMaxReconnectDelay Property	444
ServerMsgSend Property	445
ServerNames Property	446
ServerReadTimeout Property	446
ServerStartDelay Property	447
ServerStartMaxTimeout Property	447
ServerStartMaxTries Property	448
ServerWriteTimeout Property	448
Status Property	449
Subjects Property	450
Timeout Property	451
UniqueSubject Property	452
User Property	452
XtSource Property	453
TipcSrvConnCache Class	454
TipcSrvConnCache Methods	455
GMDFinish Method	455
GMDFlush Method	455
GMDInit Method	456
Init Method	457
RequestMsgs Method	457
RequestStart Method	458
RequestStop Method	459
TipcSrvConnCache Properties	460
GroupNames Property	460
LbMode Property	460
RequestTimeout Property	461
TipcSrvMon Class	463
TipcSrvMon Class Methods	466
ClientBufferGetWatch Method	466
ClientBufferPoll Method	467
ClientBufferSetWatch Method	468
ClientCbPoll Method	469
ClientCongestionGetWatch Method	470
ClientCongestionSetWatch Method	471
ClientCpuPoll Method	472
ClientExtPoll Method	473
ClientGeneralPoll Method	474
ClientInfoPoll Method	475
ClientMsgRecvGetWatch Method	477
ClientMsgRecvSetWatch Method	478
ClientMsgSendGetWatch Method	479
ClientMsgSendSetWatch Method	480
ClientMsgTrafficPoll Method	481

ClientMsgTypePoll Method	482
ClientNamesGetWatch Method	483
ClientNamesNumPoll Method	484
ClientNamesPoll Method	484
ClientNamesSetWatch Method	485
ClientOptionPoll Method	486
ClientSubjectPoll Method	487
ClientSubscribeGetWatch Method	489
ClientSubscribeNumPoll Method	490
ClientSubscribePoll Method	491
ClientSubscribeSetWatch Method	492
Create Method	493
Destroy Method	493
ProjectNamesGetWatch Method	494
ProjectNamesPoll Method	495
ProjectNamesSetWatch Method	496
ServerBufferPoll Method	496
ServerCongestionGetWatch Method	498
ServerCongestionSetWatch Method	499
ServerConnGetWatch Method	500
ServerConnPoll Method	500
ServerConnSetWatch Method	502
ServerCpuPoll Method	502
ServerGeneralPoll Method	503
ServerMaxClientLicensesGetWatch Method	505
ServerMaxClientLicensesSetWatch Method	506
ServerMsgTrafficPoll Method	507
ServerNamesGetWatch Method	508
ServerNamesPoll Method	508
ServerNamesSetWatch Method	509
ServerOptionPoll Method	510
ServerRoutePoll Method	511
ServerTimePoll Method	512
SubjectNamesGetWatch Method	513
SubjectNamesPoll Method	514
SubjectNamesSetWatch Method	515
SubscribeGetWatch Method	515
SubscribePoll Method	516
SubscribeSetWatch Method	517
TipcSrvMonExt Class	519

TipcSrvMonExt Class Methods	520
Create Method	520
Delete Method	520
Destroy Method	521
SetBinary Method	521
SetBool Method	522
SetBoolArray Method	522
SetInt2 Method	523
SetInt2Array Method	524
SetInt4 Method	524
SetInt4Array Method	525
SetInt8 Method	525
SetInt8Array Method	526
SetReal4 Method	527
SetReal4Array Method	527
SetReal8 Method	528
SetReal8Array Method	528
SetReal16 Method	529
SetReal16Array Method	529
SetStr Method	530
SetStrArray Method	531
SetUtf8 Method	531
SetUtf8Array Method	532
TipcSubjects Class	533
TipcSubjects Class Methods	534
Add Method	534
Remove Method	535
RemoveAll Method	535
TipcSubjects Class Properties	536
Count Property	536
Item Property	536
TipcSvc Class	538
TipcSvc Class Methods	539
CreateOption Method	539
GetEnv Method	539
GetOption Method	540
MsgCreate Method	541
MtCreate Method	542
MtLookup Method	543
MtLookupByNum Method	543
ParseFile Method	544
ParseStr Method	545
PutEnv Method	545
SetOption Method	546

TipcDispatcherCreate Method	546
TipcDispatcherCreateDetached Method	547
TipcSrvConnCreate Method	547
TipcSrvConnCreateNamed Method	548
TMsgFileCreate Method	548
XmlCreate Method	549
TipcSvc Class Properties	550
DefaultMsgPriority Property	550
DefaultSubjectPrefix Property	551
EnableControlMsgs Property	551
LastErrorNum Property	552
LastErrorStr Property	552
MonitorScope Property	553
ServerAutoFlushSize Property	553
ServerDeliveryTimeout Property	554
ServerKeepAliveTimeout Property	554
ServerMsgSend Property	555
ServerReadTimeout Property	556
ServerStartDelay Property	556
ServerStartMaxTries Property	557
ServerStartTimeout Property	557
ServerWriteTimeout Property	558
TipcXml Class	559
TipcXml Class Methods	560
Clone Method	560
Create Method	560
Destroy Method	560
TipcXml Class Properties	562
XmlString Property	562
TMsgFile Class	563
TMsgFile Class Methods	564
Create Method	564
Destroy Method	564
Read Method	565
Write Method	565
COM Error Handling	567
Error Handling Example	567
Index	569

Figures

Figure 1	References Window	3
Figure 2	Components Window	4
Figure 3	References Window	5
Figure 4	Object Window	7
Figure 5	Import ActiveX Control Window	8
Figure 6	OLE Custom Controls	10
Figure 7	Programmable Objects	11
Figure 8	Form Window	16
Figure 9	Code Window	16
Figure 10	Property Pages Window	17
Figure 11	Current View of Code Window	18
Figure 12	Window Details	24
Figure 13	PowerBuilder Window	27
Figure 14	SSChat Login Window	28
Figure 15	The Display Board Form Window	38
Figure 16	Form Window	48
Figure 17	Code Window	48
Figure 18	Current View of Code Window	50
Figure 19	Current View of Code Window	51
Figure 20	Example of Your Program	53
Figure 21	SSChat Login Window	54

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

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

TIBCO 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 guide provides the detailed information you need to understand and use ActiveX and COM with TIBCO SmartSockets. Before using this guide, 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*.

This guide is meant for Windows users only. ActiveX and COM are supported only on the Windows platform.

Topics

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

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 ActiveX and COM Programmer's Guide*
- *TIBCO SmartSockets API Quick Reference*
- *TIBCO SmartSockets Application Programming Interface*
- *TIBCO SmartSockets C++ User's Guide*
- *TIBCO SmartSockets C++ 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) reference 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.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

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.



The TIBCO SmartSockets™ COM and ActiveX APIs have been deprecated and will be removed in a future version.

This chapter provides instructions for using the TIBCO SmartSockets ActiveX Components and the TIBCO SmartSockets COM Library to write distributed applications using TIBCO SmartSockets. For information about installing TIBCO SmartSockets, the SmartSockets ActiveX Components, and the SmartSockets COM Library, refer to the *TIBCO SmartSockets Installation Guide*. For general SmartSockets concepts that apply to all implementations, see the *TIBCO SmartSockets User's Guide*. A basic understanding of the Component Object Model (COM) and ActiveX is essential to understand the material presented in this guide. This background material is not covered here and is beyond the scope of this guide.

Topics

- *Loading the TIBCO SmartSockets ActiveX Components, page 3*
- *Loading the TIBCO SmartSockets COM Library, page 5*
- *TIBCO SmartSockets ActiveX Control Icons, page 6*
- *Using Delphi Version 5 with ActiveX, page 8*
- *Using PowerBuilder with ActiveX, page 10*

Overview

You can develop fully graphical SmartSockets distributed applications with either of these:

- SmartSockets ActiveX Components in Visual Basic, Visual C++, Power Builder, Delphi, and other environments that support ActiveX.
- SmartSockets COM Library in Visual Basic and C++.

Note that in some cases, the scope of material covered in this guide is different for the SmartSockets ActiveX Components than it is for the SmartSockets COM Library. This is due to the fact that the SmartSockets ActiveX Components provide controls while the SmartSockets COM Library does not.

This guide focuses on using the Microsoft Visual Basic development environment, as that is the most widely used development environment. Note that SmartSockets ActiveX Components and the SmartSockets COM Library are only supported on Windows platforms. SmartSockets releases on UNIX platforms are shipped without the SmartSockets ActiveX Components or the SmartSockets COM Library.

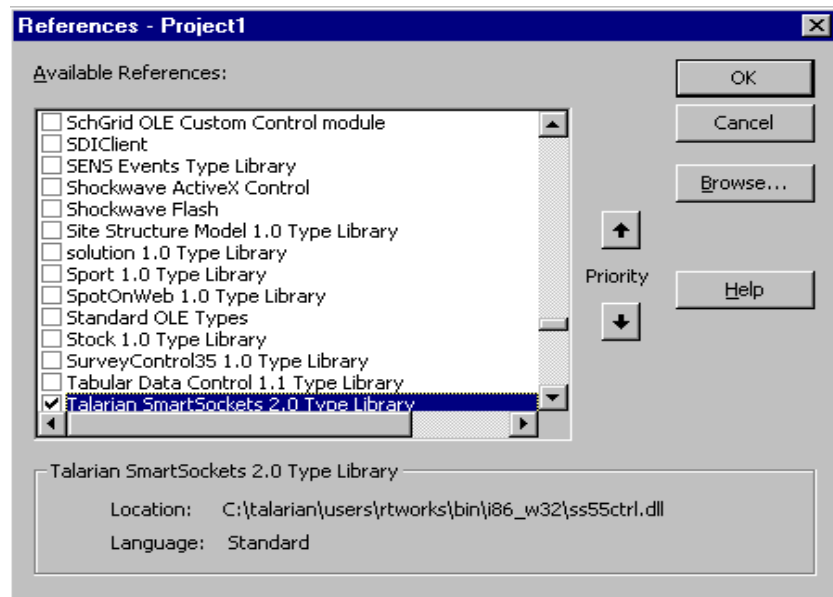
For the SmartSockets ActiveX Components, examples are also presented using the context of Delphi and PowerBuilder, but the main body of the documentation assumes a Visual Basic environment. For more information on translating examples to other ActiveX-enabled environments, consult the documentation supplied with those environments.

Loading the TIBCO SmartSockets ActiveX Components

To load the TIBCO SmartSockets ActiveX Components into your Visual Basic environment, follow these steps:

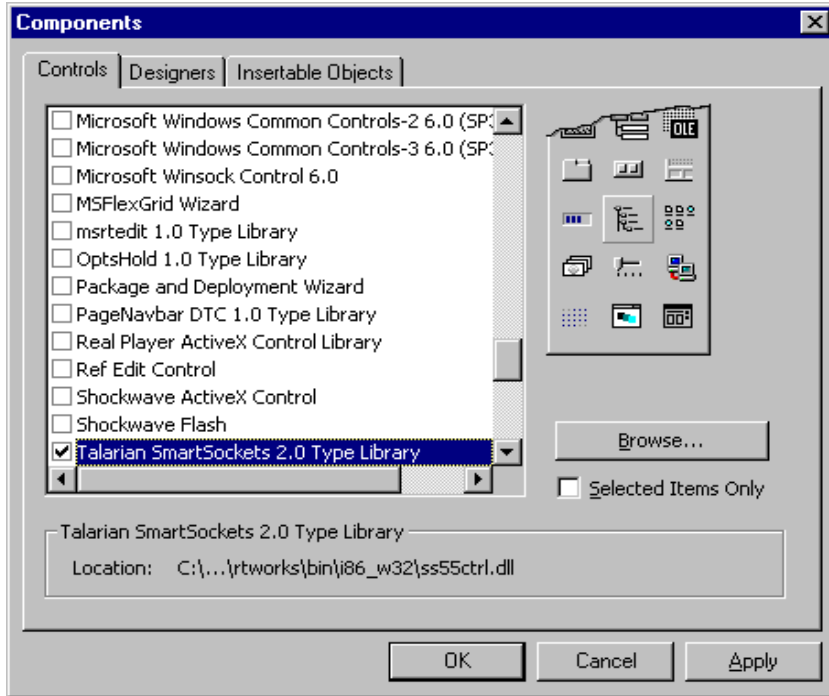
1. Select References from the Project pull-down menu. The References window appears, as shown in Figure 1. The References window allows you to select another application's objects to be available in your code by setting a reference to that application's object library. In this case, TIBCO SmartSockets is made available by setting a reference to the SmartSockets 2.0 Type Library.

Figure 1 References Window



2. Using the scroll bar, select SmartSockets 2.0 Type Library and click OK.
3. Select Components from the Project pull-down menu. The Components window appears, as shown in Figure 2.

Figure 2 Components Window



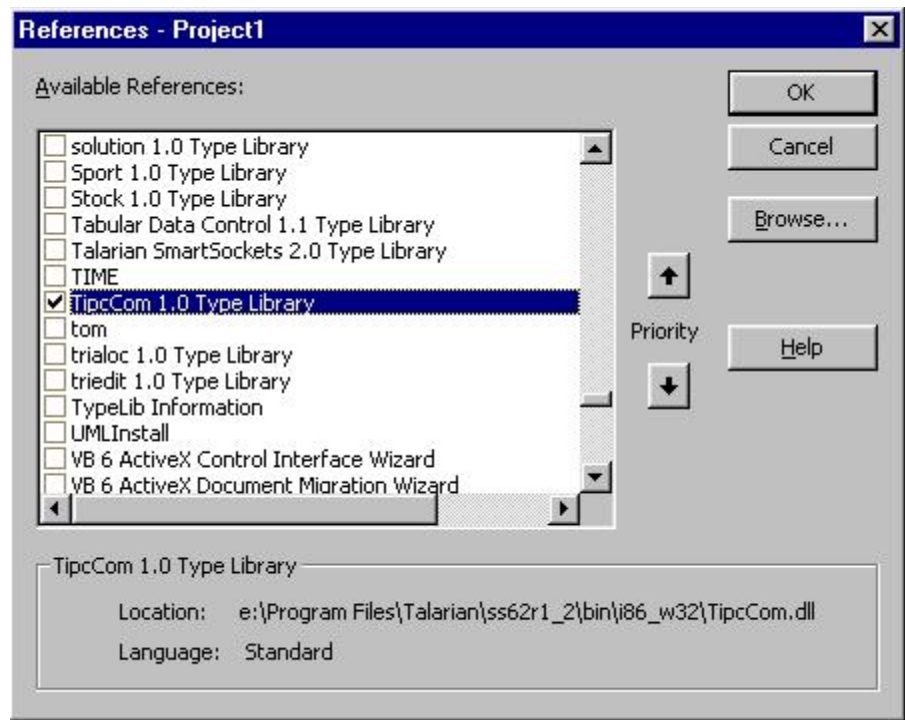
4. Using the scroll bar, select SmartSockets 2.0 Type Library and click OK. Four SmartSockets ActiveX controls appear in the Controls Palette: TipcSvc, TipcErrorCb, TipcProcessCb and TipcMon controls. You can see illustrations of these icons in Figure 7.

Loading the TIBCO SmartSockets COM Library

To load the TIBCO SmartSockets COM Library into your Visual Basic environment, follow these steps:

1. Select References from the Project pull-down menu. The References window appears, as shown in Figure 3. The References window allows you to select another application's objects to be available in your code by setting a reference to that application's object library. In this case, TIBCO SmartSockets is made available by setting a reference to the TipcCom 1.0 Type Library.

Figure 3 References Window



2. Using the scroll bar, select TipcCom 1.0 Type Library and click OK.

TIBCO SmartSockets ActiveX Control Icons

The ActiveX control icons added to the Visual Basic General Controls Palette are:

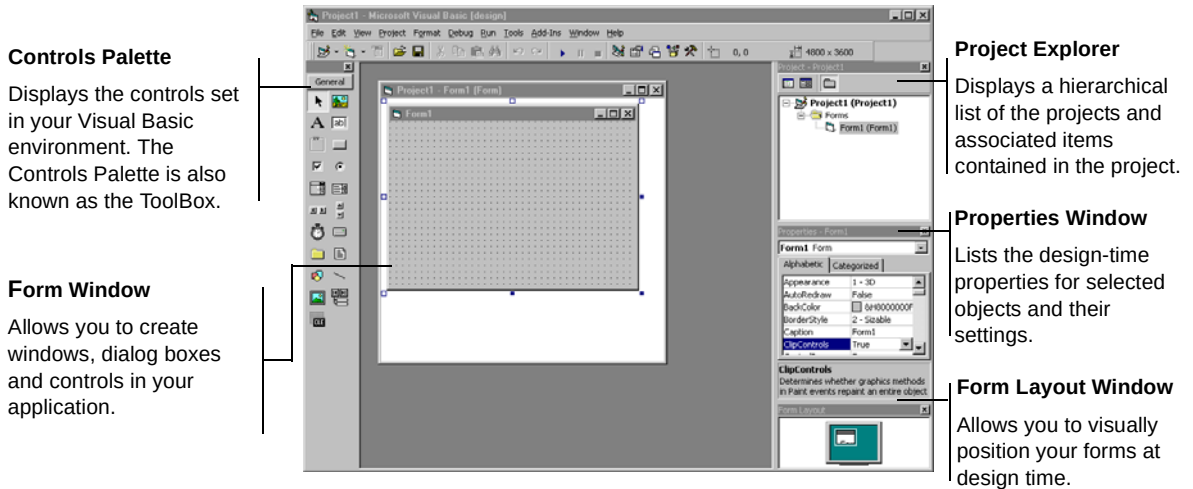
Table 1 ActiveX Control Icons

Icon	Control	Description
	TipcSvc	Provides an interface to the options used to connect to RTserver, as well as sending and receiving messages to and from RTserver. Use only one TipcSvc control per Code Form.
	TipcProcessCb	Provides an interface for handling SmartSockets messages arriving to your Integrated Development Environment (IDE) program from an event. You may use multiple TipcProcessCB controls on a Code Form.
	TipcErrorCb	Provides an interface for handling SmartSockets errors. Use only one TipcErrorCb per Code Form.
	TipcMon	Provides an interface for monitoring SmartSockets applications. Use only one TipcMon per Code Form.

To get started with the SmartSockets ActiveX components, using the Visual Basic development environment, follow these steps:

1. Start Visual Basic. The New Project window appears.
2. Select Standard EXE and click Open (or double-click the Standard EXE icon). The Object window appears, as shown in Figure 4. To access this window during design time, select Object from the View pull-down menu.

Figure 4 Object Window



Configuring Your Visual Basic Environment for ActiveX

To permanently add the SmartSockets ActiveX components to your Visual Basic environment, follow these steps:

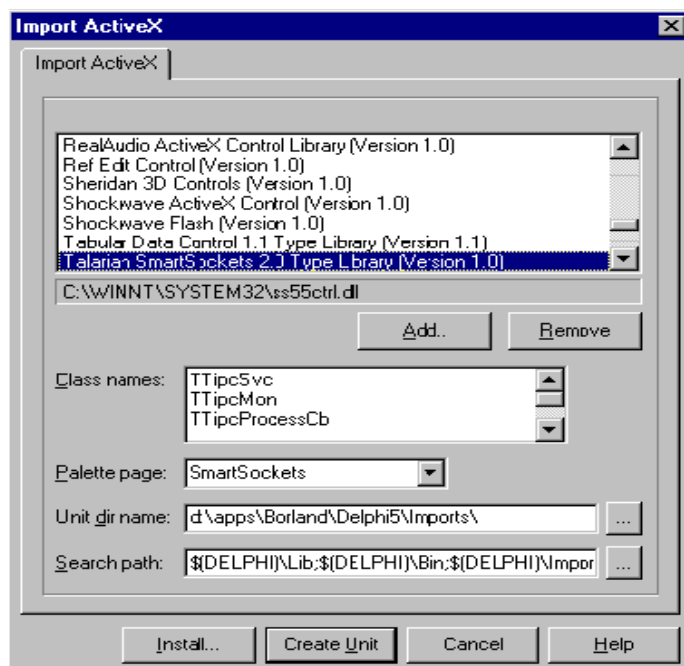
1. Right-click in the Controls Palette area to display the Palette menu. Select Add Tab. The New Tab Name window appears.
2. Type SMRTSOCK and click OK. The SMRTSOCK tab appears at the bottom of the Controls Palette.
3. Select each of the SmartSockets control icons and drag them into the SMRTSOCK tab you just created. To display and access the SmartSockets Controls, select the SMRTSOCK tab displayed at the bottom of the General Controls palette.

Using Delphi Version 5 with ActiveX

To use the SmartSockets ActiveX Components with Delphi Version 5, follow these steps:

1. Start Delphi.
2. Select Import ActiveX Control from the Component pull-down menu as shown in Figure 5.

Figure 5 Import ActiveX Control Window



3. Using the scroll bar, select SmartSockets ActiveX Controls and click Install.... If the SmartSockets ActiveX Controls do not appear, click Add and add the file ss55ctrl.dll by browsing to its install location.
4. Install the SmartSockets ActiveX Controls into either a new package or an existing package and click OK.

If you are using an older version of Delphi, you may need to:

1. Select Install Packages from the Component pull-down menu and check the box for the package you just imported, and click OK.
2. Select Import Type Library from the Project pull-down menu.

3. Using the scroll bar, select the SmartSockets Type Library and click OK.

You can verify that the SmartSockets ActiveX controls are installed for Delphi if you see the SmartSockets icons (as shown in Table 1 on page 6) in the Component Palette under the ActiveX Tab.

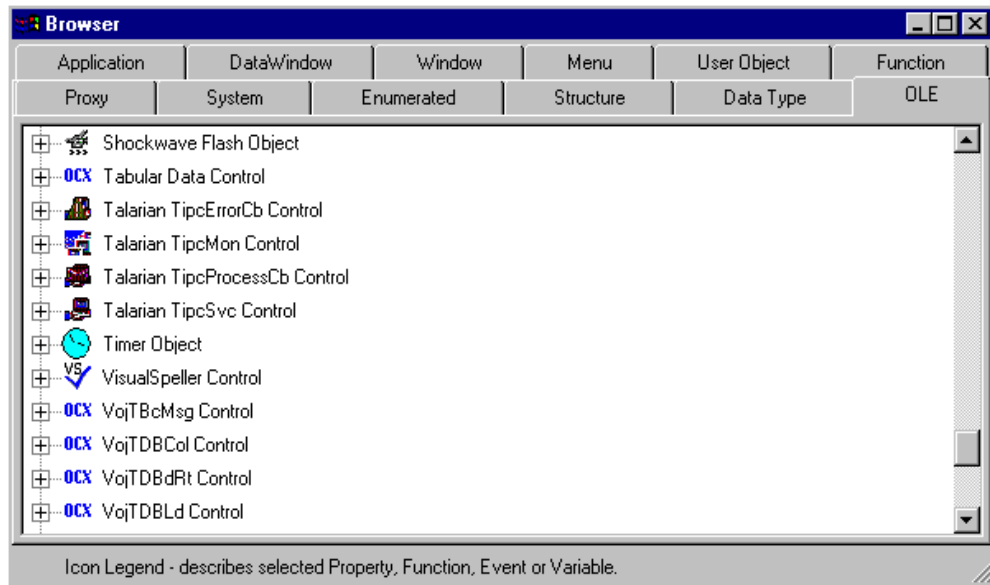
Using PowerBuilder with ActiveX

To use the SmartSockets ActiveX Components with PowerBuilder, you must be familiar with PowerBuilder Version 6.0.

You can browse the OLE controls to verify that the SmartSockets ActiveX controls are installed correctly. This section describes how to browse the OLE controls.

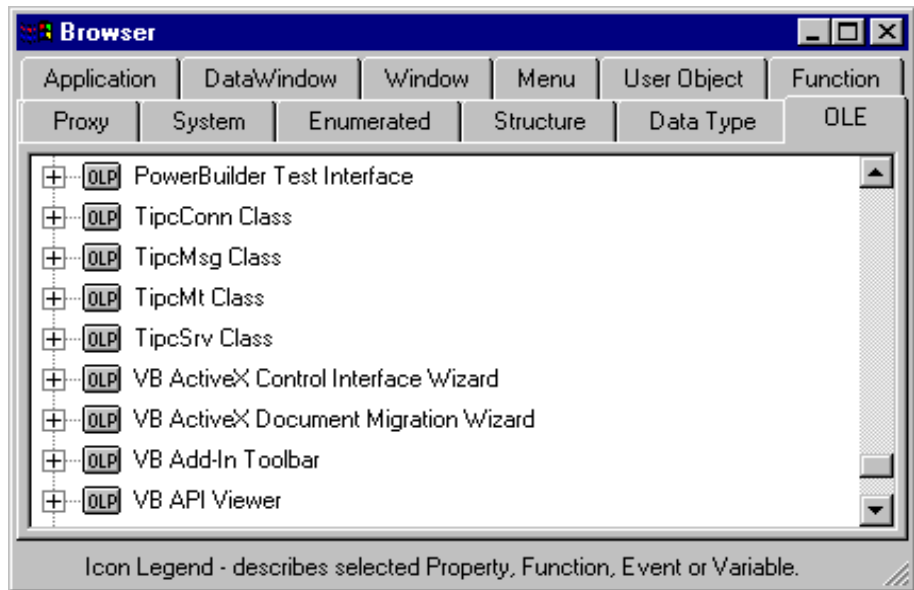
1. Start PowerBuilder.
2. Click Browser on the PowerBar.
3. Select the OLE tab in the Browser window.
4. Double click the OLE Custom Controls to see a list of all registered OLE controls, as shown in Figure 6. Scroll through the list and verify that the four Controls are present.

Figure 6 OLE Custom Controls



5. Double click the Programmable Objects to see a list of all registered objects in PowerBuilder, as shown in Figure 7. Scroll through the list to see the Tipc* classes.

Figure 7 Programmable Objects



See Chapter 2, Using ActiveX Components for detailed information about the SmartSockets ActiveX classes.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Chapter 2 Using ActiveX Components

In this chapter you write, compile and execute a sample Visual Basic program that uses SmartSockets for interprocess communication. At the end of this chapter, you will walk through two Visual Basic example programs that are included with your SmartSockets ActiveX Components distribution.

While these manuals are oriented towards the Visual Basic development environment, examples are repeated for Delphi and PowerBuilder as well. For more information on those development environments, see the specific vendor-supplied documentation for those environments. For more information on SmartSockets concepts, see the *TIBCO SmartSockets User's Guide*.

Topics

- *Getting Started, page 14*
- *Writing a TIBCO SmartSockets ActiveX Program in Visual Basic, page 15*
- *Writing a TIBCO SmartSockets ActiveX Program in Delphi Version 5, page 21*
- *Writing a TIBCO SmartSockets ActiveX Program in PowerBuilder, page 23*
- *Example Programs, page 27*

Getting Started

Before you begin writing your first program, be sure that SmartSockets and the SmartSockets ActiveX Components are installed properly on your system. Refer to the *TIBCO SmartSockets Installation Guide* for information about installing SmartSockets and the SmartSockets ActiveX Components.

TIBCO SmartSockets ActiveX Classes

SmartSockets messaging applications are referred to as RTclients. RTclients connect to a publish and subscribe message router referred to as RTserver. The SmartSockets ActiveX controls provide full RTserver interaction through the TipcSvc and TipcSrv classes. Peer-to-peer connections are not supported for ActiveX RTclients.

The TipcSvc class provides an interface to an application's RTserver connection, enabling you to set options and manage the connection to RTserver. The TipcSrv class manages your application's messaging tasks, such as sending messages to RTserver and processing messages. TipcSrv also controls an application's subscriptions.

The TipcMsg class is used to create and destroy messages, add and retrieve message data fields, as well as set a number of options for each message.

Other ActiveX classes are the TipcMt class for managing message types, the TipcMsgFields class for manipulating the fields of SmartSockets messages, the TipcSubjects class for containing and maintaining subject subscriptions, and the TipcErrorCb and TipcProcessCb callback event classes. Chapter 4, ActiveX Controls Reference provides reference information about these classes.

The most commonly used SmartSockets ActiveX classes are:

TipcErrorCb	Handles errors that result from network problems or from problems with connections to the RTserver.
TipcMon	Manages monitoring of RTclient applications through polling or watching.
TipcMsg	Creates and destroys messages, adds and retrieves message data fields, as well as sets a number of per-message options.
TipcProcessCb	Handles messages arriving to your IDE program from an event.
TipcSrv	Manages the messaging tasks your application performs when connected to an RTserver.
TipcSvc	Provides an interface and manages the connection to an RTserver, enabling options to be set.

Writing a TIBCO SmartSockets ActiveX Program in Visual Basic

Now you can write a SmartSockets program in Visual Basic. The first part of your SmartSockets program invokes a connection to an RTserver and sets up the callbacks that will process messages and errors from the server. The second part sends and receives a message to and from a subscribing RTclient.

Table 2 displays the icons referenced in this chapter. The SmartSockets ActiveX Control icons are shown in Table 1 in Chapter 1, ActiveX and COM Environments

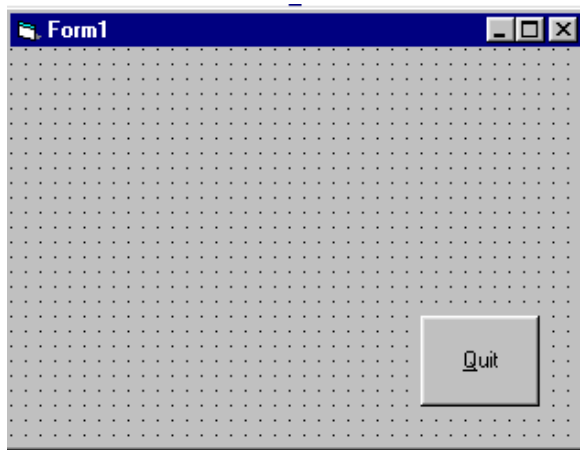
Table 2 Icons

Icon	Icon Name	Description
	Command	Located within the General Controls Palette and is used to create buttons on forms.
	Start	Located on the toolbar and is used to start the program.
	TextBox	Located within the General Controls Palette and is used to create text on forms.
	TEdit	Located within the Standard tab on the Components palette and is used to edit text.

Perform these steps to create the RTserver connection program:

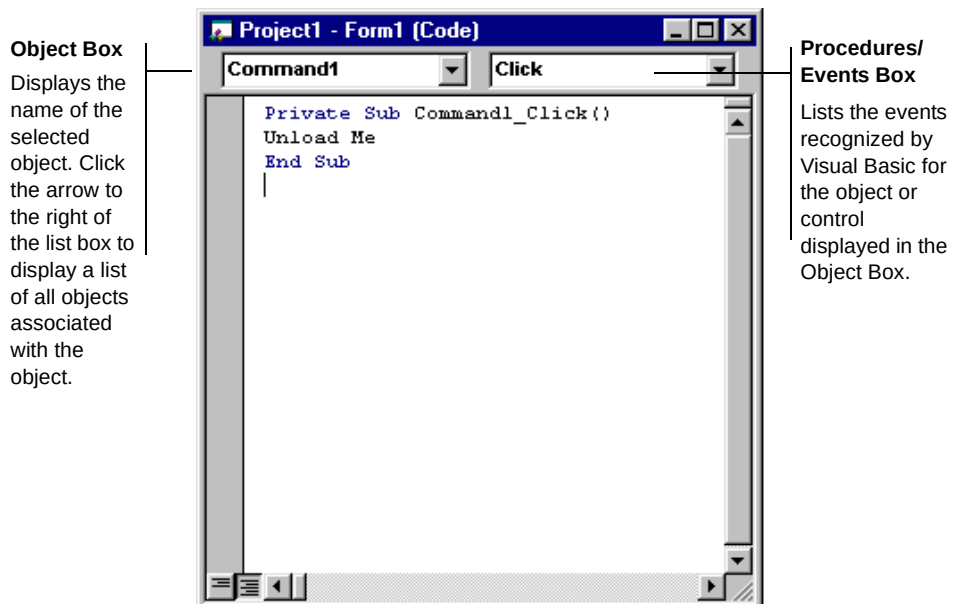
1. From the General Controls Palette, select the Command icon and draw a button in the Form window, as pictured in Figure 8.

Figure 8 Form Window



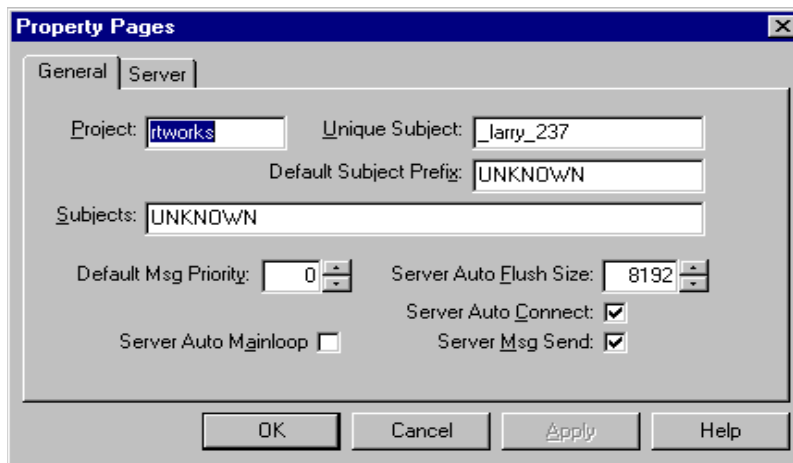
2. In the Properties window, to the right of the Caption field, type: Quit. The button changes in real-time, reflecting what you type—the button should now have the label Quit.
3. Double-click the Quit button you just created. The Code window appears, as shown in Figure 9.

Figure 9 Code Window



4. At the current insertion point in the Code window, type: `unload me`, indenting just as you would when writing other programs.
5. Close the Code window to return to the Form window. (To test the program, click the Start icon from the toolbar. A window appears displaying the Quit button. Click Quit to close and return to the Code window.)
6. From the SMRTSOCK palette, select the TipcSvc icon and draw a box in the Form window. This creates the TipcSrv object on the Form.
7. In the Properties window, TipcSvc1 is displayed in the Name field. Replace TipcSvc1 with Tipc in the Name field. This step is merely to demonstrate how you can change the name of an object.
8. Left click the Custom field in the Properties window (the Ellipsis icon appears to the right of the field). Click the Ellipsis icon to show the Property Pages window, as shown in Figure 10.

Figure 10 Property Pages Window



9. In the Subjects field, remove the word UNKNOWN, type: `hail`, then click Apply. Note that the Subjects field in the Properties window now displays "hail." Click OK to close the Property Pages window.
10. Double-click on an empty portion of the Form to display the Form1_Load Code window. (You can access this general code window any time during design time by selecting Code from the View pull-down menu.)
11. At the current insertion point, type:

```
Set Srv = Tipc.Srv
If Srv.ConnStatus <> 2 Then
    Srv.Create
End If
```



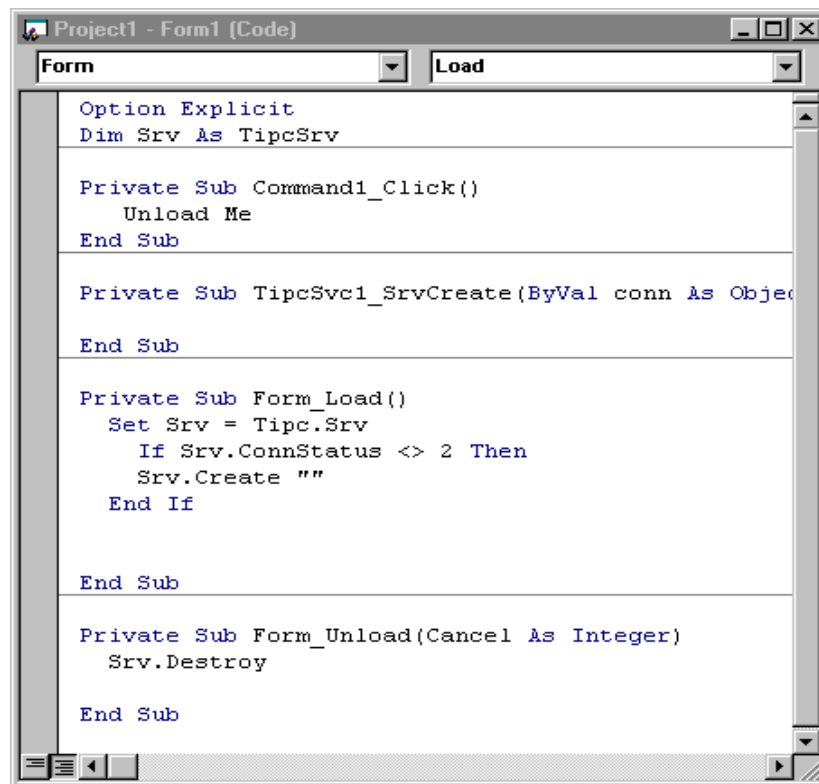
After you type the period, following the TipcSrv object, a pull-down menu appears, showing the applicable properties and methods for this object.

12. At the top of the Code window, click the arrow next to the Form pull-down menu to display the options. The options are: (General), Command1, Form, and Tipc (the Form option is currently highlighted). Select (General) and at the insertion point, type:

```
Option Explicit
Dim Srv As TipcSrv
```

13. Click in the Procedures/Events box, shown in Figure 9, scroll down and select Unload. In the Code window, a new subroutine called PrivateSubForm_Unload now appears.
14. At the current insertion point, type: Srv.Destroy. Figure 11 is an example of your program. Close the Code window.

Figure 11 Current View of Code Window



15. From the SMRTSOCK Controls palette, select the TipcProcessCb icon. Draw a box in the Form window to create a TipcProcessCb object (for processing messages from another client).
16. In the Properties window, type: InfoCb in the Name field.
17. From the SMRTSOCK Controls palette, select the TipcProcessCb icon. Draw a box in the Form window.
18. In the Properties window, type: SubjCb in the Name field.
19. In the Form window, double-click the TipcSvc icon. The Code window appears.

20. At the current insertion point, type the following code below the line that specifies the call to Tipc_SrvCreate:

```
If old_conn_status = 0 Then
    Dim mt As TipcMt
    Srv.StdSubjectSubscribe True, False
    Set mt = Tipc.MtLookup("info")
    Srv.AddProcessCb InfoCb.Object, mt
    Srv.AddSubjectCb SubjCb.Object, "/hail", mt
End If
```

21. While the SmartSockets Command prompt window is visible, run the program again to see the list of subscribers for the Subject named hail. This also establishes and destroys the RTserver connection. Click Quit to end the program. See Testing the TIBCO SmartSockets ActiveX Program for instructions on testing the SmartSockets program.
22. Close the Code window.
23. From the General Controls palette, select the TextBox icon and draw a box in the Form window.
24. While the TextBox is selected in the Form window, in the Properties window change the Width value to 2555.
25. From the Form window, double-click the TipcProcessCb icon for the InfoCb object. The Code window appears.
26. At the current insertion point, type these lines of code below the line that specifies the InfoCb_TipcMsg call:

```
Dim tmsg As TipcMsg
Set tmsg = msg.Clone
tmsg.Dest=tmsg.Sender
tmsg.NumFields=0
tmsg.Fields.Add "Message Received!"
Srv.MsgSend tmsg
Srv.Flush
tmsg.Destroy
```

27. Close the Code window.

28. From the Form window, double-click the TipcProcessCb icon for the SubjCb object. The Code window appears.
29. At the current insertion point, type the following lines of code below the line that specifies the SubjCb_TipcMsg call:

```
Dim tmsg As TipcMsg
Set tmsg = msg
Text1.Text = tmsg.Fields (0)
```

30. Close the Code window.
31. Select the TipcSvc icon in the Form window and change the AutoMainLoop value to True in the Properties window. This enables automatic processing of SmartSockets messages.
32. While both the Visual Basic and the SmartSockets Command Prompt windows are visible, start the Visual Basic program. Notice that there is now a Text box displayed above the Quit button in the Form window. Do not click the Quit button.

Testing the TIBCO SmartSockets ActiveX Program

To test the SmartSockets program, perform these steps:

1. Select **Programs > SmartSockets > Command Prompt** from the Windows Start menu to open a SmartSockets Command Prompt window.
2. To invoke RTmon, type this command at the SmartSockets Command prompt:
`C:\Program Files\tibco\ss68>rtmon -runtime`
3. Type the following command at the RTmon prompt to invoke watching, which displays monitoring information:
`MON> watch client_names`
4. RTserver is initialized. Now type the following command to monitor the RTclient that is subscribing to the "hail" subject:
`MON> watch Subject_Subscribe hail`
5. To receive and display the watch responses, type the following command at the RTmon prompt:
`MON> run`
6. Arrange your windows so that you can view the SmartSockets and Visual Basic windows simultaneously.
7. Run your Visual Basic program again by pressing the Start arrow key located in the toolbar. Notice in the SmartSockets Command Prompt window that when you run the program, a connection is established and when you quit the program, the RTclient connection is destroyed.

8. While the Visual Basic program is running, type the following commands at the RTmon prompt (to display “Hello World” in the client window). (**Note:** To stop run mode, type: Ctrl-c from the RTmon Runtime command prompt):

```
MON> send info hail "Hello World!"
MON> run
```

Writing a TIBCO SmartSockets ActiveX Program in Delphi Version 5

Now you are going to write a SmartSockets program in Delphi (see Using Delphi Version 5 with ActiveX on page 8 for information on running Delphi). The first part of your SmartSockets program invokes a connection to an RTserver and the second part sends and receives a message to and from a subscribing RTclient. Perform these steps to write the program:

1. From the Additional tab on the Component palette, double-click the BitBtn icon. The BitBtn component is added to the form.
2. In the Object Inspector window, select the Properties tab, change the Kind property to bkClose, and then press the Enter key. The button’s caption changes to Close.
3. From the ActiveX tab on the Component palette, double-click the TipcSvc icon. The TipcSvc component is added to the form.
4. In the Object Inspector window, TipcSvc1 is displayed in the Name field on the Properties tab. Replace TipcSvc1 with Tipc. This step is merely to demonstrate how you can change the name of an object.
5. Select **View > Code Explorer** to open the Code Explorer window. Expand the Classes folder and double-click the TForm1 class definition. Under private, add: Srv: ITipcSrv;
6. Double-click on an empty portion of the Form to create the FormCreate method and display the Code window.
7. At the current insertion point, type:


```
Srv := ITipcSrv(Tipc.Srv);
If Srv.ConnStatus <> 2 Then
Begin
    Srv.Create;
End
```
8. Double-click the Close button of the form to create the BitBtn1Click method.
9. At the current insertion point, type:

```
Srv.Destroy;
```

10. From the ActiveX tab on the Component palette, double-click the TipcProcessCb icon.
11. In the Object Inspector window, select the Properties tab, change the Name property from TipcProcessCb1 to SubjCb, and press the Enter key.
12. In the Form window, select the TipcSvc icon. In the Object Inspector window, select the Events tab, set the OnSrvCreate property to SrvCreateCb, and press the Enter key.
13. Add a var section to the SrvCreateCb procedure and type:

```
mt: ITipcMt;
```

After the begin statement, type:

```
If old_conn_status = 0 Then
  Begin
    mt := ITipcMt(Tipc.MtLookup('info'));
    Srv.AddSubjectCb(IUnknown(SubjCb.OleObject), 'hail',
      mt, 0, 'User Cb Arg');
    Srv.Subjects.Add('hail', False);
    Srv.Flush();
  End
```

14. From the Standard tab on the Components palette, double-click the TEdit icon. In the Object Inspector window, select the Properties tab, set the Text property to a blank and press the Enter key.
15. Select the TipcProcessCb icon on the Form window for the SubjMsg object. In the Object Inspector window, select the Events tab, set the OnTipcMsg event to SubjMsgCb and press the Enter key.
16. Add a var section to the SubjMsgCb method and type: tmsg: ITipcMsg;. After the begin statement, type:


```
tmsg := ITipcMsg(msg);
tmsg.Current := 0;
Edit1.Text := tmsg.NextStr;
tmsg.Dest := tmsg.Sender;
tmsg.NumFields := 0;
tmsg.AppendStr('Message Received!' + String (param));
Srv.MsgSend(tmsg);
Srv.Flush;
```
17. While the TipcSvc icon in the Form window is selected, change the AutoMainLoop value to True on the Properties tab in the Object Inspector window. This enables automatic processing of SmartSockets messages.
18. Run the program.

Testing the TIBCO SmartSockets ActiveX Program

To test the SmartSockets program, perform these steps:

1. At the Windows Start menu, open a SmartSockets Command Prompt window, by selecting **Programs > SmartSockets > Command Prompt**.
2. To invoke RTmon, type the following command at the SmartSockets command prompt:

```
C:\Program Files\tibco\ss68>rtmon -runtime
```

3. While Delphi is running, type the following commands at the RTmon prompt: (**Note:** You may need to stop the run mode with Ctrl-c.)

```
MON> send info hail "Hello World!"
MON> run
```

This is an example of text displayed in the RTmon window:

```
sender = </_workstation_153>
sending server = </_workstation_175>
dest = </_workstation_170>
max = 2048
size = 40
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 = 1>:
str "Message Received!"
```

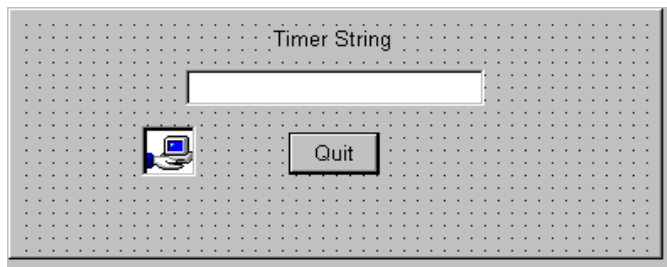
Writing a TIBCO SmartSockets ActiveX Program in PowerBuilder

Now you will write a SmartSockets program in PowerBuilder. The first part of your SmartSockets program invokes a connection to an RTserver and the second part of the program sends and receives a message to and from a subscribing RTclient.

1. Click Application on the PowerBar.
2. Click New on the PainterBar. Type: sstutor.pb1 in the File name field and click Save. Type: sstutor in the Application name field and click OK. Click No when prompted to generate an application template.
3. Click Window on the PowerBar. Click New to generate a new window. Resize the window to make it smaller.

4. Click Properties on the PainterBar to change the window properties. Type: SmartSockets Tutorial for the title. Click OK to close the Properties window.
5. Click Button to add a command button to the window. When you click the down arrow next to this button, a pull-down menu shows all controls that can be inserted into the view. Click anywhere inside the window to position the command button control. Display the Property window by selecting Properties on the PainterBar. Type: cb_quit in the Name box to change the name of the control. Type quit in the Text box. Select the Default checkbox. Click OK.
6. Select **Button > Static Text** control and click inside the window to position this control. Display the Property window by selecting Properties on the PainterBar. Type: st_timer in the Name box to change the name of control. Click OK to close the Properties window.
7. Select **Static Text > Single Line Edit** control and click inside the window to position this control. Enlarge the control so that it is wider. Display the Property window by selecting Properties on the PainterBar. To change the name of the control, type: sle_input in the Name box. Click OK to close the Properties window.
8. Select **Single Line Edit > OLE > Insert Control tab > Tibco TipcSvc control > OK**. Click inside the window to position this control. Display the Property window by selecting Properties on the PainterBar. Type: ole_tipc in the Name box to change the name of the control. Click the OLE Control Properties button to bring up the Property window of the TipcSvc control. In the Subjects field, type: hail, then click OK. Click OK to close the Property window. Figure 12 is an example of what the window you created will look like.

Figure 12 Window Details



9. To create a function to handle the processing of an INFO message, select **Declare>Window Functions** from the Menu Bar. Click New and type: `uf_process_info` in the Name box. Type: `msg` in the Name box inside the Arguments box. Select Any in the Type listbox. Click OK. Type these lines of code:

```
oleobject info_msg = msg
sle_input.text = info_msg.fields(0)
info_msg.dest = info_msg.sender
info_msg.NumFields = 0
info_msg.AppendStr("Message Received!")
ole_tipc.object.srv.MsgSend(info_msg)
ole_tipc.object.srv.Flush
return 0
```

Click in the background of the window to select the window. Click Script. Make sure that you are looking at the script for the Open event for the window. The title of the Script window should read:

open for (Untitled) returns long

Type these lines of code:

```
oleobject srv
srv = ole_tipc.object.srv
If srv.ConnStatus <> 2 Then
    srv.Create
End If
timer(1) // Calls the timer event every second
st_timer.text = String(Now(), "hh:mm:ss")
```

10. Select the Close event for the window by opening the Select Event pull-down menu. PowerBuilder automatically compiles the script when you close the PowerScript Painter or change events. Type this line of code:

```
ole_tipc.object.srv.destroy
```

11. Open the Select Event pull-down menu and select the Timer event for the window. Type these lines of code:

```
oleobject srv, msg, mt
string mt_name
integer READ = 0

srv = ole_tipc.object.srv
st_timer.text = String(Now(), "hh:mm:ss")

if srv.Check(READ, 0.0) then
    msg = srv.MsgNext(0.0)
    if msg.IsNull then
        return
    end if
    mt = msg.type
    mt_name = mt.name
    Choose Case mt_name
        CASE "info"
            uf_process_info(msg)
        CASE else
            MessageBox("Msg Received", mt_name)
    end Choose
```

```

        srv.MsgProcess(msg)
    msg.Destroy
end if

```

12. Click Return to return to the Window Painter. PowerBuilder automatically compiles the script when you close the PowerScript Painter.
13. Click the Quit command button and then click Script to open the PowerScript Painter. Make sure that you are looking at the clicked event for the cb_quit object. Type this code:

```

ole_tipc.object.srv.destroy
HALT

```

14. Click Return on PainterBar to save and compile the script.
15. Select the RTserver control, ole_tipc, and then click Script to open the PowerScript Painter. Make sure that you are looking at the srvcreate event for the ole_tipc object. If not, then change the event by choosing Select Event from the pull-down menu. Type this code:

```

if old_conn_status = 0 then
    ole_tipc.object.srv.StdSubjectSubscribe(TRUE, FALSE)
end if
HALT

```

16. Click Return on PainterBar to save and compile the script.
17. Click Save on the PainterBar to save the window. Type: w_main for the window name.
18. Click Close on the PainterBar to stop the Window Painter.
19. Click Script on the PainterBar. Make sure you are looking at the open event for the application. Type this code:

```

open(w_main)

```

20. Click Return to save and compile the script.
21. Click Save to save changes.
22. Click Run on the PowerBar to run your program. While PowerBuilder is running, type the following commands at the RTmon prompt to display “hail chief” in the client window. (**Note:** To stop run mode, type: Ctrl-c from the RTmon Runtime command prompt):

```

MON> send info hail "hail chief"
MON> run

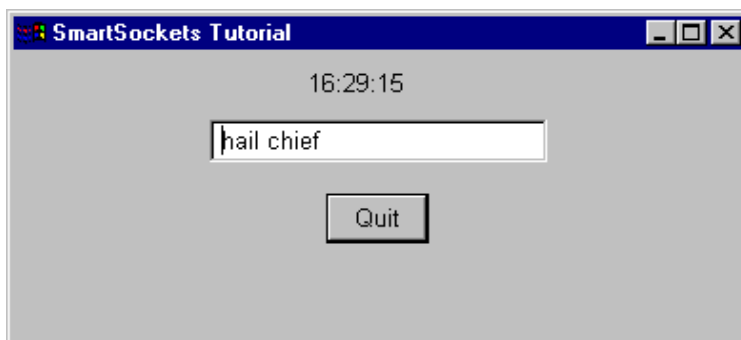
```

Figure 13 is an example of what the window will look like when it is running.



The PowerBuilder compiler does not know the correct syntax for accessing properties and functions of an ActiveX control. Therefore, it cannot check any syntax after the object property. This leaves the application open to runtime errors if the properties and functions are misnamed.

Figure 13 PowerBuilder Window



Example Programs

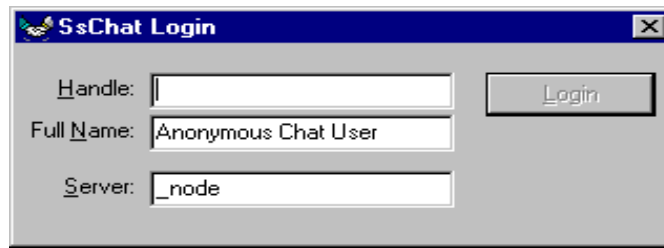
Now that you have written a simple SmartSockets program, this section presents several programs that use some of the more advanced SmartSockets features. These programs are not presented in a step-by-step instructional format, rather portions of the Visual Basic code for each program are reviewed and key elements are discussed.

SSChat: Multi-User Chat Room

The SSChat program is an RTclient that implements a simple real-time, multi-user chat room. The example files are located in the installation directory under `Examples\ActiveX\sschat`. The SSChat program is written using the SmartSockets publish-subscribe technology to implement this function with a minimum amount of code. The entire SSChat program is under 200 lines of Visual Basic code.

The SSChat application starts with the Login window, as shown in Figure 14. The Login window is a standard Visual Basic form named `frmLogin`. It allows the user to input their full name and handle, and specify to which RTServer to connect.

Figure 14 SSChat Login Window



The essential details of SSChat's implementation are included within the main form, `frmMain`.

Exploring `frmMain`

There are several form-level global variables, found under (General) (Declarations), defined as:

```
Dim mtHi As TipcMt
Dim mtBye As TipcMt
Dim msgHi As TipcMsg
Dim msgOut As TipcMsg
Dim RTserver As TipcSrv
```

The first two variables, `mtHi` and `mtBye`, hold `TipcMt` objects specifying two of the custom message types SSChat defines. The second two, `msgHi` and `msgOut`, hold pre-built messages that are sent multiple times during an execution of SSChat. Finally, `RTserver` acts as a shortcut to the `TipcSrv` object held in the `TipcSvc` component on the form.

Looking at the `Form_Load` method shows the actions taken upon entering the chat room. The custom message types are registered and the server name is configured. A connection to `RTserver` is created, and a "Hi" (`mtHi`) message is constructed, containing the user's name and chat handle.


```

Private Sub Form_Load()
    Dim btn As Integer

    kScriptHeightOffset = Height - ctlScript.Height
    kScriptMarginOffset = Width - ctlScript.RightMargin
    kScriptWidthOffset = Width - ctlScript.Width
    kMessageTopOffset = Height - ctlMessage.Top
    kMessageWidthOffset = Width - ctlMessage.Width
    ctlStatus.Panels(1) = fmLogin.ctlHandle
    ctlStatus.Panels(2) = fmLogin.ctlName

    ctlScript.SelColor = QBColor(1)
    ctlScript.SelText = "[ " + str(Now) + " ]"
    ctlScript.SelColor = QBColor(0)

    Set mtHi = ctlTipc.MtCreate("hi", 1, "str str int2")
    Set mtBye = ctlTipc.MtCreate("bye", 2, "str")

    Set msgOut = _

    ctlTipc.MsgCreate(ctlTipc.MtLookup("string_data"))
    msgOut.Dest = "_all"
    msgOut.Fields.Add fmLogin.ctlHandle.Text

    Set msgHi = msgOut.Clone
    msgHi.Type = mtHi
    msgHi.Fields.Add fmLogin.ctlName.Text
    msgHi.Fields.Add Int(True)

    Set RTserver = ctlTipc.Srv
    If Len(fmLogin.ctlServer) > 0 Then
        ctlTipc.ServerNames = fmLogin.ctlServer
    End If
    ctlTipc.UniqueSubject = fmLogin.ctlHandle
    On Error GoTo ErrorHandler
    RTserver.Create
    On Error GoTo 0
    Exit Sub

ErrorHandler:
    Unload Me
    btn = MsgBox(Err.Description)
End Sub

```



By cloning the msgOut message, the Dest property is the same for msgHi. Appending True requests other SSChat clients to reply upon receipt of the message.

Upon connection to an RTserver (through `RTserver.Create` in the code above), the `SrvCreate` event belonging to the form's `TipcSvc` object is called. This function registers the message-processing callbacks and sends the "Hi" message constructed in `Form_Load`, as shown here:

```
Private Sub ctlTipc_SrvCreate(ByVal conn As Object, ByVal _
    old_conn_status As Integer, ByVal new_conn_status As _
    Integer)
    If old_conn_status = tipcSrvConnNone Then
        RTserver.StdSubjectSubscribe True, False
        RTserver.AddProcessCb ctlHiCb.Object, mtHi, 1
        RTserver.AddProcessCb ctlByeCb.Object, mtBye, 1
        RTserver.AddProcessCb ctlDataCb.Object, _
            ctlTipc.MtLookup("string_data"), 1
        RTserver.MsgSend msgHi
        RTserver.Flush
    End If
End Sub
```

Once the `SrvCreate` code is executed, `SSChat` is in its primary mode, waiting for keystrokes from the user or messages from `RTserver`. Examine what happens when a key is pressed. Note that when you press the Enter key, the current text is sent out to the other chat participants.

```
Private Sub ctlMessage_KeyDown(KeyCode As Integer, Shift _
    As Integer)
    Dim str As String

    If (Shift And vbShiftMask) = 0 Then
        If KeyCode = vbKeyReturn Then
            msgOut.NumFields = 1
            str = ctlMessage
            If Right(str, 1) = Chr(13) Then
                str = Left(str, Len(str) - 2)
            End If
            msgOut.Fields.Add str
            RTserver.MsgSend msgOut
            RTserver.Flush
            ctlMessage.Text = ""
        End If
    End If
End Sub
```

Notes:

- Setting the NumFields property to 1 eliminates all but the first field of the message.
- msgOut is already mostly constructed; all you have to do is add the chat text, send and flush to guarantee immediate delivery.

Now that you have reviewed the sending process of chat text from SSChat, take a look at the three message processing events that handle the receiving process. The first ctlHiCb's TipcMsg event receives the "Hi" messages:



If the message was not sent from our own process, and a reply was requested (Fields(2) was True), send the originator a "Hi" message back so they know who else is in the chat room.

```
Private Sub ctlHiCb_TipcMsg(ByVal conn As Object, ByVal _
msg As Object, ByVal param As Variant)
    Dim msgIn As TipcMsg

    Set msgIn = msg
    If msgIn.Fields(0) <> msgOut.Fields(0) Then
        If msgIn.Fields(2) Then
            msgHi.Dest = msgIn.Sender
            msgHi.NumFields = msgHi.NumFields - 1
            msgHi.Fields.Add Int(False)
            RTserver.MsgSend msgHi
            RTserver.Flush
        End If
    End If
    ctlScript.SelStart = 2000000000
    ctlScript.SelRTF = "{\par\par}"
    ctlScript.SelColor = QBColor(1)
    ctlScript.SelText = "[ " + msgIn.Fields(1) + _
        "connected as " + msgIn.Fields(0) + " ]"
    ctlScript.SelColor = QBColor(0)
    ctlScript.SelStart = 2000000000
End Sub
```

Now look at the code that handles the "Bye" messages, sent when a user leaves the chat room, terminating the SSChat process. It's much the same as the "Hi" message event handler, except it does not send any replies.

```
Private Sub ctlByeCb_TipcMsg(ByVal conn As Object, ByVal _
msg As Object, ByVal param As Variant)
    Dim msgIn As TipcMsg

    Set msgIn = msg
    ctlScript.SelStart = 2000000000
    ctlScript.SelRTF = "{\par\par}"
    ctlScript.SelColor = QBColor(1)
    ctlScript.SelText = "[ " + msgIn.Fields(0) + _
        "disconnected ]"
    ctlScript.SelColor = QBColor(0)
    ctlScript.SelStart = 2000000000
End Sub
```

The event handler for chat data messages is shown below (similar to the other two message events, it updates the output window with the originating chat user's handle and chat text):

```
Private Sub ctlDataCb_TipcMsg(ByVal conn As Object, ByVal _
    msg As Object, ByVal param As Variant)
    Dim msgIn As TipcMsg

    Set msgIn = msg
    ctlScript.SelStart = 2000000000
    ctlScript.SelRTF = "{\par\par}"
    ctlScript.SelBold = True
    ctlScript.SelText = msgIn.Fields(0) + ": "
    ctlScript.SelBold = False
    ctlScript.SelText = msgIn.Fields(1)
    ctlScript.SelStart = 2000000000
End Sub
```

In the code above, `Fields(0)` is the chat handle and `Fields(1)` is the message. Note that the `TipcMsg`'s `NextStr` method could also have been used to access these values.

Finally, `SSChat`'s `Form_Unload` subroutine that sends the "Bye" message and disconnects from `RTserver` is shown below:



It checks to see if the `RTclient` is connected. If so, it disconnects, otherwise it exits.

```
Private Sub Form_Unload(Cancel As Integer)
    Dim msgBye As TipcMsg
    If RTserver.ConnStatus = tipcSrvConnFull Then
        Set msgBye = ctlTipc.MsgCreate(mtBye)
        msgBye.Dest = msgOut.Dest
        msgBye.Fields.Add msgOut.Fields(0)
        RTserver.MsgSend msgBye
        RTserver.Destroy
    End If
End Sub
```

Data Flow

The data flow in the SSChat program is:

1. Chat users joining the room are announced to the group members by publishing a "Hi" message containing information about the new user.
2. Other SSChat RTclients receive the new user's announcement and reply directly and exclusively to the originating RTclient, by publishing a "Hi" message back.
3. Upon receiving a "Hi" or "Bye" message, an SSChat RTclient updates its output window with the status change indicating which other client entered or left the room.
4. As data messages are received, SSChat displays these messages to its chat window.

WhoWhere: Electronic Message Board

The WhoWhere program is a graphical RTclient, implementing an in/out message board useful for tracking employee whereabouts. The WhoWhere program is located in the installation directory under `Examples\Activex\whowhere`. WhoWhere is an electronic counterpart to the sign in and out boards commonly used in corporate offices. WhoWhere uses SmartSockets publish-subscribe technology. Other SmartSockets features demonstrated by this program are:

- Custom message types
- Process and error events
- Messages within messages
- Hierarchical naming scheme for publish-subscribe messages
- Publishing to entire subjects as well as individual RTclients

WhoWhere Overview

WhoWhere tracks employee whereabouts. Each employee belongs to a department. The department an employee belongs to specifies the SmartSockets subject used to set apart message board groups. This allows potentially thousands of users in hundreds of departments to be present on the same LAN or WAN, without causing conflict.

The WhoWhere application is composed of several Visual Basic forms and one code module. Most of the forms (those for logging in, managing the configuration and specifying leave information) are handled with the usual Visual Basic techniques; examination of the source code should be fairly self-explanatory. The important parts of the application are handled by the Display Board form and the modGlobals module.

The modGlobals Module

First, look at some of the data structures used in the modGlobals module:

```
Public Type configType
    Name As String
    Password As String
    Lunch As Integer
    Email As String
    Homepage As String
    MailApp As String
    WebApp As String
    Server As String
    Alerts As Integer
    Department As String
    EmailBrowser As Integer
End Type
Public Config As configType
```

The configType structure holds the local configuration information. This data is stored in the Windows registry by the SaveSettings subroutine and reloaded with LoadSettings. This allows the configuration to be persistent between executions of the application. The data includes the user's name, password and other configuration information. A subset of this information is maintained for all the other known employees on the message board as well, as shown in the userType data structure:

```
Public Type userType
    Who As String
    Email As String
    Homepage As String
    Message As String
    Where As String
    ReturnInfo As String
End Type

Public Myself As userType
Public Users(maxUsers) As userType
Public nUsers As Long
Public wwSubject As String

Public RTserver As TipcSrv

Public mtAnnounce As TipcMt
Public mtUpdate As TipcMt
Public mtResponse As TipcMt

Public Enum wwMessageTypes
    wwAnnounce = 100
    wwUpdate
    wwResponse
End Enum
```

Myself holds a copy of this user's message board information; Users() is the array that holds the actual message board information. nUsers is the number of users in the array and therefore the number of users displayed on the board. wwSubject holds the publish-subscribe subject name used for the WhoWhere client communication.

A global handle to the TipcSrv method used by the application is held in RTserver. Additionally, WhoWhere defines three new message types identified by the numbers 100, 101, and 102 as the enumeration wwMessageTypes specifies; mtAnnounce, mtUpdate and mtResponse act as global references to these message type objects that are created.

Next, the initializeClient subroutine sets up the SmartSockets components and the environment. Note that some of the properties set here could also have been set on the ActiveX property sheet for the component. However, setting the properties in the code aids clarity for maintenance.

```
Public Sub initializeClient()
    Dim RTc As TipcSvc
    Set RTc = frmBoard.TipcSvc1
    Set RTserver = RTc.Srv

    ' disconnect from server if we're connected
    If RTserver.ConnStatus <> tipcSrvConnNone Then
        RTserver.Destroy
    End If
    RTc.AutoMainLoop = True
    RTc.ServerNames = Config.Server
    RTc.Project = programName
    frmBoard.appStatus "Connecting to RTserver..."
    RTserver.Create ("")
    frmBoard.appStatus "Connected to " + RTserver.Node + _
        " (PID " + Format(RTserver.Pid) + ") "
End Sub
```

When RTserver.Create is executed, the SrvCreate event of the TipcSvc object is invoked. Note also that when connecting to RTserver with the Create method, the parameter is always an empty string (""). In this example, the wwSubject configuration is the publish-subscribe subject this client is interested in for message board activity. For example, if the RTclient's department is set to Marketing, the message board subject is /WhoWhere/Marketing.

```
Private Sub TipcSvc1_SrvCreate(ByVal conn As Object, ByVal _
    old_conn_status As Integer, ByVal new_conn_status As _
    Integer)
    If Debugging Then
        Debug.Print "Server connection status is now = ";
        Debug.Print Format(new_conn_status); " (was = ";
        Debug.Print Format(old_conn_status); ")"
    End If
    If new_conn_status <> tipcSrvConnFull Then
        Exit Sub ' nothing special if not a full connxn
    End If

    ' once full connection is made, set up everything
    Dim RTc As TipcSvc
    Set RTc = frmBoard.TipcSvc1
    ' create custom message types
```

```

Set mtAnnounce = RTc.MtCreate("wwAnnounce", wwAnnounce, _
    "int2 msg")
Set mtResponse = RTc.MtCreate("wwResponse", wwResponse, _
    "int2 msg")
' update is the msg sent inside announce and response,
' and also by itself for updating status
Set mtUpdate = RTc.MtCreate("wwUpdate", wwUpdate, _
    "int2 str str str str str str")

' install event handlers for message types
RTserver.AddProcessCb frmBoard.TipcAnnounce.Object, _
    mtResponse
RTserver.AddProcessCb frmBoard.TipcAnnounce.Object, _
    mtAnnounce
RTserver.AddProcessCb frmBoard.TipcUpdate.Object, _
    mtUpdate
RTserver.AddDefaultCb frmBoard.TipcDefault.Object
RTserver.AddErrorCb frmBoard.TipcErrorCb1.Object

' set up the WhoWhere pub/sub subject and subscribe
wwSubject = "/" + programName + "/" + Config.Department
RTserver.StdSubjectSubscribe True, False
RTserver.Subjects.Add wwSubject, False

' send a message announcing our arrival
doUpdate
End Sub

```

Only message board traffic published to the wwSubject subject is seen by this client. It is this hierarchical naming feature of SmartSockets that accounts for its scalability. Without changing the client, and with only one level of partitioning (the department), a large number of separate message boards can coexist.

As illustrated in the code, a call to the doUpdate subroutine is made. This sends the equivalent of a "Hello" message from this RTclient to the other users displaying this department's message board. The following code for doUpdate builds and sends an announcement message using the mtAnnounce object as a reference (notice how a second SmartSockets message of type mtUpdate is included inside the announcement message):

```

Public Sub doUpdate()
    ' build and send an announcement message
    ' for initially joining a message board subject
    ' or doing an 'update all'
    Dim RTc As TipcSvc
    Set RTc = frmBoard.TipcSvc1

    Dim am As TipcMsg, myData As TipcMsg
    Set myData = RTc.MsgCreate(mtUpdate)
    With myData
        .AppendInt2 messageFormat
        .AppendStr Myself.Who
        .AppendStr Myself.Email
        .AppendStr Myself.Homepage
        .AppendStr Myself.Where
        .AppendStr Myself.Message
        .AppendStr Myself.ReturnInfo
    End With

```



```

Set am = RTc.MsgCreate(mtAnnounce)
am.AppendInt2 messageFormat
am.AppendMsg myData
am.Dest = wwSubject
RTserver.MsgSend am
RTserver.Flush
End Sub

```

The following subroutine, `publishMyStatus`, is used to send an update message when the current user's status changes, for example, when they leave or return to the office. A message of type `mtUpdate` is sent alone this time, not included inside another message.

```

Private Sub publishMyStatus()
' build and send just an update message when
' our status changes (go to lunch, return, etc.)
Dim RTc As TipcSvc
Set RTc = frmBoard.TipcSvc1

Dim myData As TipcMsg
Set myData = RTc.MsgCreate(mtUpdate)
With myData
.AppendInt2 messageFormat
.AppendStr Myself.Who
.AppendStr Myself.Email
.AppendStr Myself.Homepage
.AppendStr Myself.Where
.AppendStr Myself.Message
.AppendStr Myself.ReturnInfo
.Dest = wwSubject
End With
RTserver.MsgSend myData
RTserver.Flush
End Sub

```

The `setAway` subroutine updates the buttons available on the `WhoWhere` graphical user interface (GUI). There are two subroutines, `goAway` and `comeBack`, that make calls to `publishMyStatus`, as shown here:

```

Public Sub goAway()
publishMyStatus
frmBoard.setAway True
End Sub

Public Sub comeBack()
Myself.Where = inStatus
publishMyStatus
frmBoard.setAway False
End Sub

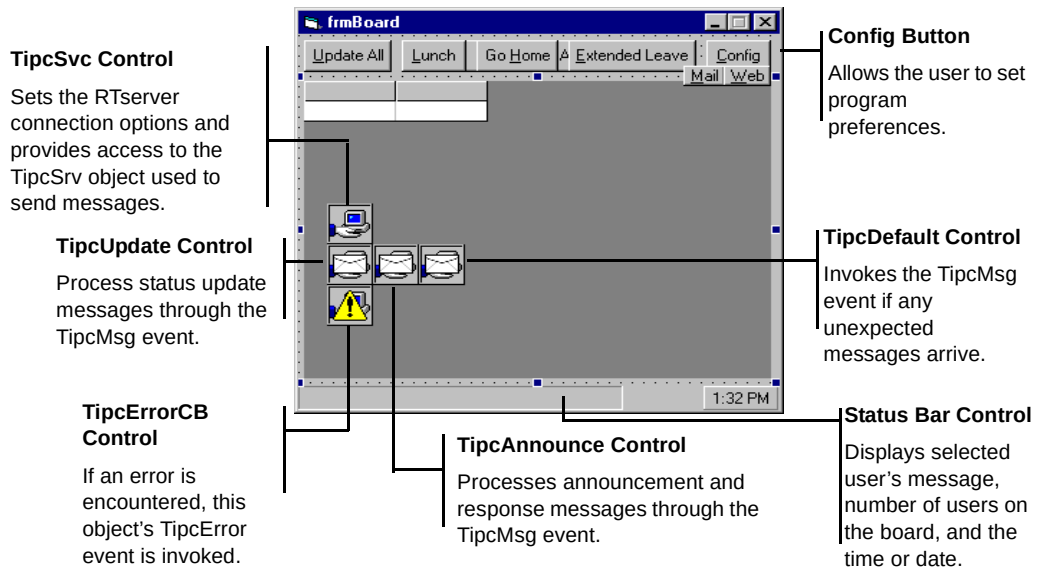
```

The Display Board Form

The Display Board Form is the main user-interface object of the WhoWhere application. This form displays the message board with dynamic updating and allows the user to interact with the program through the command buttons.

Figure 15 shows the message board Form window of the WhoWhere Visual Basic project.

Figure 15 The Display Board Form Window



In the next example, the form's Load subroutine calls the other initialization routines such as initializeGUI, which positions and sizes various screen elements. initializeClient configures the environment for SmartSockets and subscribes to the relevant subjects. The form's Unload code saves the program settings and disconnects from the RTserver.

```
Private Sub Form_Load()
    Me.Left = Val(GetSetting(programName, "WindowPos", _
        "Left", Str(Screen.Width - Me.Width)))
    Me.Top = Val(GetSetting(programName, "WindowPos", _
        "Top", "0"))
    Me.Caption = programName + " v" + program version
    initializeGUI
    Me.Visible = True
    Me.Show
    noErrBox = True
    initializeClient
End Sub
```

```

Private Sub Form_Unload(Cancel As Integer)
    SaveSetting programName, "WindowPos", "Left", _
        Str(Me.Left)
    SaveSetting programName, "WindowPos", "Top", _
        Str(Me.Top)
    saveSettings
    RTserver.Destroy
End Sub

```

As shown in the next example, the `comeBack` and `goAway` subroutines are called when the absence or return (`btnBack`) buttons are clicked:

```

Private Sub btnHome_Click()
    Myself.Where = "HOME"
    Myself.Message = "(left for the day)"
    Myself.ReturnInfo = "the next work day"
    goAway
    btnBack.SetFocus
End Sub

Private Sub btnLunch_Click()
    Myself.Where = "LUNCH"
    Myself.Message = "(at lunch)"
    Myself.ReturnInfo = "at " + Format(DateAdd("n", _
        Config.Lunch, Time))
    goAway
    btnBack.SetFocus
End Sub

Private Sub btnExtended_Click()
    frmExtended.Show 1
    If extendedCancel Then Exit Sub
    goAway
    btnBack.SetFocus
End Sub

Private Sub btnBack_Click()
    comeBack
End Sub

```

The displayBoard subroutine is the most important in terms of user-interface; it re-populates the message board with data from the Users() array. The board has nUsers+1 rows; the extra is used to display the column headings. The message board is implemented with a Microsoft MSFlexGrid ActiveX component:

```
Public Sub displayBoard()
    Dim i As Integer, wh As String

    board.Rows = nUsers + 1
    For i = 0 To nUsers - 1
        board.Row = i + 1
        board.Col = 0
        board.Text = Users(i).Who
        board.Col = 1
        wh = Users(i).Where
        If wh <> inStatus Then
            wh = wh + " - returns " + Users(i).ReturnInfo
        End If
        board.Text = wh
        board.CellAlignment = flexAlignLeftCenter
    Next i
    btnMail.Visible = False
    btnWeb.Visible = False
    board.Col = 1
    board.Row = 1
    board.ColSel = 1
    board.RowSel = board.Rows - 1
    board.Sort = flexSortStringNoCaseAsending

    If Config.Alerts Then
        Beep
    End If
End Sub
```

TipcAnnounce decodes announce and response messages, and dispatches the newUser subroutine to add a user to the message board. As shown this example, if TipcAnnounce processes an announcement message, it responds by publishing a response message directly back to the originator:

```
Private Sub TipcAnnounce_TipcMsg(ByVal conn As Object, _
    ByVal msg As Object, ByVal param As Variant)
    Dim m As TipcMsg

    Set m = msg
    If Debugging Then
        Debug.Print "(announce) received " + m.Type.Name _
            + "msg from " + m.Sender
    End If
    Dim mver As Integer
    mver = m.NextInt2
    If mver > messageFormat Then
        wrongMessageFormat "ANNOUNCE", mver
    Else
        Dim m2 As TipcMsg
        Set m2 = m.NextMsg
        mver = m2.NextInt2
        newUser m2
        ' if announcement send a response
        ' don't respond to responses
        If m.Type.Num = wwAnnounce Then
            Dim resp As TipcMsg
            Set resp = TipcSvc1.MsgCreate(mtResponse)
            resp.Dest = m.Sender
            resp.AppendInt2 msgversion
        End If
    End If
End Sub
```

```

        Dim myData As TipcMsg
        Set myData = TipcSvc1.MsgCreate(mtUpdate)
        With myData
            .AppendInt2 messageFormat
            .AppendStr Myself.Who
            .AppendStr Myself.Email
            .AppendStr Myself.Homepage
            .AppendStr Myself.Where
            .AppendStr Myself.Message
            .AppendStr Myself.ReturnInfo
        End With
        resp.AppendMsg myData
        TipcSvc1.Srv.MsgSend resp
        TipcSvc1.Srv.Flush
    End If
End If
End Sub

```

TipcDefault handles unexpected messages received by the WhoWhere RTclient. In this example program, a message box appears, notifying the user of the alert:

```

Private Sub TipcDefault_TipcMsg(ByVal conn As Object, _
ByVal msg As Object, ByVal param As Variant)
    Dim m As TipcMsg
    Set m = msg
    MsgBox "Received unexpected message of type " + _
        m.Type.Name
End Sub

```

The TipcErrorCb callback's TipcError event is invoked when a SmartSockets error occurs. In this case, the relevant error information is displayed to the user to acknowledge:

```

Private Sub TipcErrorCb1_TipcError(ByVal conn As Object, _
ByVal c_err_num As Long, ByVal os_err_num As Long, ByVal _
socket_err_num As Long, ByVal smart_sockets_err_num _
As Long)
    MsgBox "There has been a SmartSockets error!" + _
        Chr(10) + "C Error: " + Str(c_err_num) + Chr(10) + _
        "OS Error: " + Str(os_err_num) + Chr(10) + _
        "Socket Error: " + Str(socket_err_num), _
        vbExclamation + vbApplicationModal + _
        vbOKOnly, "SmartSockets Error" + _
        Str(smart_sockets_err_num)
End Sub

```

The TipcSvc1_SrvDestroy event is invoked when the connection to RTserver is terminated. Here it does nothing unless the global variable (see modGlobals) Debugging is set to a non-zero value:

```

Private Sub TipcSvc1_SrvDestroy(ByVal conn As Object, _
ByVal old_conn_status As Integer, ByVal new_conn_status As _
Integer)
    If Debugging Then
        Debug.Print "Disconnected from server (old status =";_
            old_conn_status; ")"
    End If
End Sub

```

The `newUser` subroutine takes an update message as a parameter, and adds the user information contained within to the `Users()` array, first removing any old instance of the user. In this example, the number of users is incremented and `displayBoard` is called to refresh the form display:

```
Private Sub newUser(um As TipcMsg)
    Dim thisname As String
    thisname = um.NextStr

    ' if user already on board, remove them
    Dim i As Integer, wasRemoved As Boolean
    i = 0
    While (i < nUsers And Not wasRemoved)
        If Users(i).Who = thisname Then
            For j = i To nUsers - 2
                Users(j) = Users(j + 1)
            Next j
            nUsers = nUsers - 1
            wasRemoved = True
        End If
        i = i + 1
    Wend

    With Users(nUsers)
        .Who = thisname
        .Email = um.NextStr
        .Homepage = um.NextStr
        .Where = um.NextStr
        .Message = um.NextStr
        .ReturnInfo = um.NextStr
    End With
    nUsers = nUsers + 1
    displayBoard
    showCount
End Sub
```

As shown in the next example, the `updateUser` subroutine takes an update message as a parameter, and updates the user information contained within to the `Users()` array. If the user is not currently in the array, they are added. Like the `newUser` subroutine, it calls `displayBoard` to refresh the form display.

```
Private Sub updateUser(um As TipcMsg)
    Dim thisname As String, uIndex As Integer
    thisname = um.NextStr

    ' user already on board?
    uIndex = -1
    For i = 0 To nUsers - 1
        If Users(i).Who = thisname Then
            uIndex = i
            Exit For
        End If
    Next i

    ' add this user if necessary
    If uIndex = -1 Then
        uIndex = nUsers
        nUsers = nUsers + 1
    End If

    With Users(uIndex)
```

```

        .Who = thisname
        .Email = um.NextStr
        .Homepage = um.NextStr
        .Where = um.NextStr
        .Message = um.NextStr
        .ReturnInfo = um.NextStr
    End With
    displayBoard
    showCount
End Sub

```

The `btnUpdate` subroutine, as shown below, is invoked when the Update All button is clicked. It resets the user count and calls `doUpdate`, re-issuing an announce message. The other users' message board applications will see this and send response messages directly to the running WhoWhere RTclient, populating the `Users()` array as the message callback events are invoked.

```

Private Sub btnUpdate_Click()
    nUsers = 0
    doUpdate
    board.SetFocus
End Sub

```

Data Flow

The flow of messages used in the WhoWhere program is:

1. Users just joining the message board group (specified by the department configuration field) are announced to the group members by publishing an `mtAnnounce` message containing information specific to the new user.
2. WhoWhere RTclients receive the new user's announcement and reply directly and exclusively to the originating RTclient by publishing an `mtResponse` message containing their information.
3. Upon receiving an `mtResponse` message, a WhoWhere RTclient adds the enclosed user information to their message board.
4. All WhoWhere RTclients receive `mtUpdate` messages, processing and changing the updated information to their message board in real-time.
5. If you click Update All, the board information is discarded and re-built by starting over as if a new client subscribed to the message board.

Notes

There are some points to note when you examine all the WhoWhere source code:

- Although not shown here, the `frmConfiguration` form calls `initializeClient` upon successful completion (that is, when you click OK). This ensures that any changes made to the configuration are accurately reflected by the WhoWhere application's state.
- When WhoWhere is invoked for the first time, the Configuration dialog box appears where you set up your message board environment. Once you enter the information, the Log In window appears.
- There is message format version-control in the WhoWhere program. The first field in every message sent is added with a line similar to:

```
message.AppendInt2 messageFormat
```

This value is decoded with passages similar to the next example in the message processing events:

```
Dim mver As Integer
mver = message.NextInt2
If mver > messageFormat Then
    wrongMessageFormat "MESSAGETYPE", mver
Else
    ' process the message
End If
```

The code extracts the first field from the messages, a two-byte integer, and checks it against the global constant `messageFormat` (see `modGlobals` for the definition of `messageFormat`). This ensures that if newer versions of WhoWhere, with different message grammars are present in the same department, the older RTclients do not corrupt their data with incompatible messages. Your applications may need more sophisticated message version control. This can be implemented with the `UserProp` property of `TipcMsg` objects.

- Remember, placing break points in the source code and stepping through the code with Step Into (the F8 key) is the way to see the sequence of events being encountered by the program.
- The global constant `Debugging` (see `modGlobals`) can be set to a non-zero value. This enables printing of diagnostic information in the Visual Basic Immediate window, which is useful for grasping the overall operation of the WhoWhere RTclient.
- When you enter your name in the Name field of the Configuration window, enter your first name, followed by the last name (for example, Jane Smith). Names are automatically displayed on the message board in alphabetical order: last name appearing first, followed by a comma and then the first name (for example, Smith, Jane).

Chapter 3 **Using the COM Library**

In this chapter you write, compile and execute a sample Visual Basic program that uses SmartSockets for interprocess communication. At the end of this chapter, you will walk through two Visual Basic example programs that are included with your SmartSockets COM Library distribution. For more information on SmartSockets concepts, see the *TIBCO SmartSockets User's Guide*.

Topics

- *Getting Started, page 46*
- *Writing a TIBCO SmartSockets COM Program in Visual Basic, page 47*
- *Example Programs, page 54*

Getting Started

Before you begin writing your first program, be sure that SmartSockets and the SmartSockets COM Library are installed properly on your system. Refer to the *TIBCO SmartSockets Installation Guide* for information about installing SmartSockets and the SmartSockets COM Library.

TIBCO SmartSockets COM Classes

SmartSockets messaging applications are referred to as RTclients. RTclients connect to a publish and subscribe message router referred to as RTserver. The SmartSockets COM Library provides full RTserver interaction through the TipcSvc and TipcSrvConn classes. Peer-to-peer connections are not supported for COM RTclients.

The TipcSvc class is a convenience interface for creating and managing other classes.

The TipcMsg class is used to create messages, add and retrieve message data fields, as well as set a number of options for each message.

Other COM classes are the TipcMt class for managing message types, the TipcMsgFields class for manipulating the fields of SmartSockets messages, the TipcSubjects class for containing and maintaining subject subscriptions, and the TipcErrorCb and TipcProcessCb callback event classes. Chapter 5, COM Library Reference provides reference information about these classes.

The most commonly used SmartSockets COM classes are:

TipcErrorCb	Handles errors that result from network problems or from problems with connections to the RTserver.
TipcSrvMon	Manages monitoring of RTclient applications through polling or watching.
TipcMsg	Creates and destroys messages, adds and retrieves message data fields, as well as sets a number of per-message options.
TipcProcessCb	Handles messages arriving to your program from an event.
TipcSrvConn	Manages a connection to RTserver and messaging tasks your application performs once connected to an RTserver.
TipcSvc	Provides an interface for creating objects and enables options to be set.

Writing a TIBCO SmartSockets COM Program in Visual Basic

Now you can write a SmartSockets program in Visual Basic. The first part of your SmartSockets program invokes a connection to an RTserver and sets up the callbacks that will process messages and errors from the server. The second part sends and receives a message to and from a subscribing RTclient.

Table 3 displays the icons referenced in this chapter.

Table 3 Icons

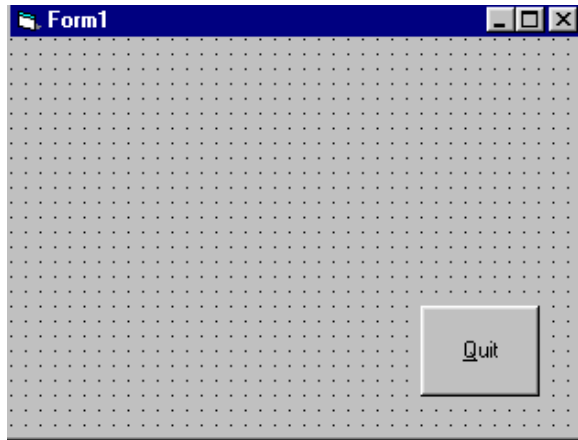
Icon	Icon Name	Description
	Command	Located within the General Controls Palette and is used to create buttons on forms.
	Start	Located on the toolbar and is used to start the program.
	TextBox	Located within the General Controls Palette and is used to create text on forms.
	TEdit	Located within the Standard tab on the Components palette and is used to edit text.

Perform these steps to create the RTserver connection program:

1. From the General Controls Palette, select the Command icon and draw a button in the Form window, as pictured in Figure 16.

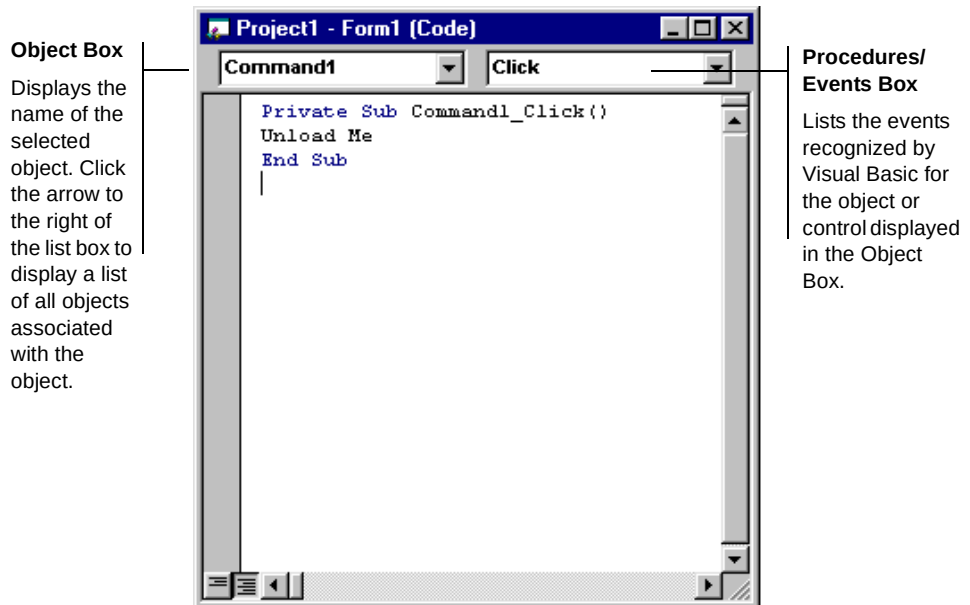
ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Figure 16 Form Window



2. In the Properties window, to the right of the Caption field, type: Quit. The button changes in real-time, reflecting what you type—the button should now have the label Quit.
3. Double-click the Quit button you just created. The Code window appears, as shown in Figure 17.

Figure 17 Code Window



4. At the current insertion point in the Code window, type: `unload me`, indenting just as you would when writing other programs.
5. Close the Code window to return to the Form window. (To test the program, click the Start icon from the toolbar. A window appears displaying the Quit button. Click Quit to close and return to the Code window).
6. At the top of the Code window, click the arrow next to the Form pull-down menu to display the options. The options are: (General), Command1, and Form. Select (General) and at the insertion point, type:

```
Option Explicit
Dim srv_conn As TipcSrvConn
Dim Tsvc As New TipcSvc
Dim terminate As Boolean
```

7. From the menu, select **Project > Add Class Module**. Select Class Module. A new code window opens. At the top of the code window, type:

```
Implements TipcProcessCb
```

8. Next, select `TipcProcessCb` from the pull-down menu on the top left of the code window. This generates a subroutine that looks like the following:

```
Private Sub TipcProcessCb_TipcMsg(ByVal conn As Object, ByVal pMsg
As TIPCCOMLib.ITipcMsg, ByVal arg As Variant)
```

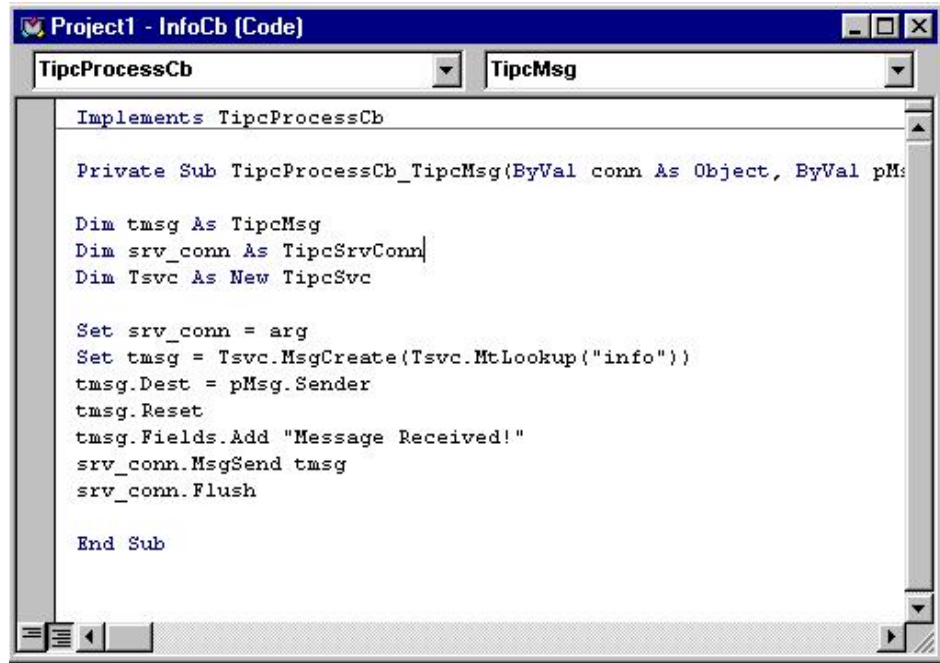
```
End Sub
```

9. Enter the following code into the `TipcProcessCb_TipcMsg` subroutine:

```
Dim tmsg As TipcMsg
Dim srv_conn As TipcSrvConn
Dim Tsvc As New TipcSvc
Set srv_conn = arg
Set tmsg = Tsvc.MsgCreate(Tsvc.MtLookup("info"))
tmsg.Dest = pMsg.Sender
tmsg.Reset
tmsg.Fields.Add "Message Received!"
srv_conn.MsgSend tmsg
srv_conn.Flush
```

10. In the properties of the class module, change the name from Class1 to InfoCb.
The code window should look as follows:

Figure 18 Current View of Code Window



11. Close the code window.
12. From the menu, select **Project > Add Class Module**. Select Class Module. A new code window opens. At the top of the code window, type:

```
Implements TipcProcessCb
```

13. Select TipcProcessCb from the pull-down menu on the top left of the code window. This generates a subroutine that looks like the following:

```

Private Sub TipcProcessCb_TipcMsg(ByVal conn As Object, ByVal pMsg As TIPCCOMLib.ITipcMsg, ByVal arg As Variant)

End Sub

```

14. Enter the following code into the TipcProcessCb_TipcMsg subroutine:

```

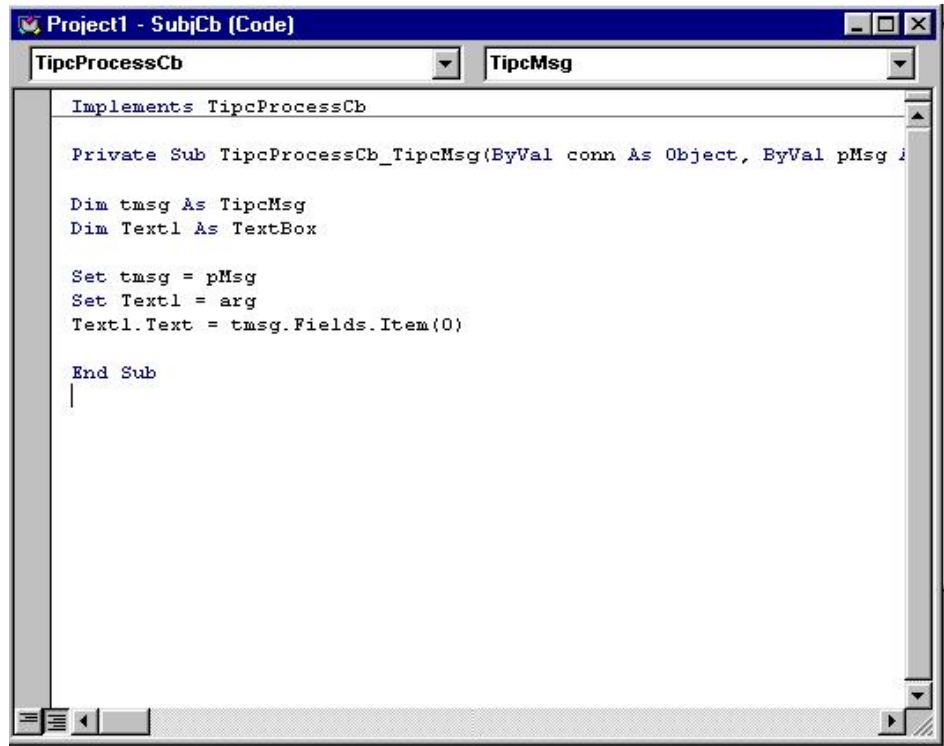
Dim tmsg As TipcMsg
Dim Text1 As TextBox

Set tmsg = pMsg
Set Text1 = arg
Text1.Text = tmsg.Fields.Item(0)

```

15. In the properties of the class module, change the name from Class1 to SubjCb.
The code window should look as follows:

Figure 19 Current View of Code Window



16. Close the code window.
17. Double-click on an empty portion of the Form to display the Form1_Load Code window.



You can access this general code window any time during design time by selecting Code from the View pull-down menu.

18. At the current insertion point, type:

```

Dim info_cb As New InfoCb
Dim subject_cb As New SubjCb
Dim mt As TipcMt

On Error GoTo Error_Handler:

Set srv_conn = Tsvc.TipcSrvConnCreate("/unique_subject", "rtworks", "tcp:_node", "")
srv_conn.Open

```

```

Set mt = Tsvc.MtLookup("info")
srv_conn.SubjectSetSubscribe "/hail", True
srv_conn.ProcessCbCreate info_cb, mt, 1, srv_conn
srv_conn.SubjectCbCreate subject_cb, "/hail", mt, 1, Text1
srv_conn.Flush

```

```
Exit Sub
```

```

Error_Handler:
MsgBox Err.Description, vbOKOnly, "Connection Error"

```

19. Click in the Procedures/Events box, shown in Figure 17, scroll down and select Activate. In the Code window, a new subroutine called `PrivateSubForm_Activate` appears.

20. At the current insertion point, type:

```

terminate = False
While terminate = False
    srv_conn.MainLoop (0.25)
    DoEvents
Wend

srv_conn.Close

```

21. Click in the Procedures/Events box, shown in Figure 17, scroll down and select Unload. In the Code window, a new subroutine called `PrivateSubForm_Unload` appears.

22. At the current insertion point, type:

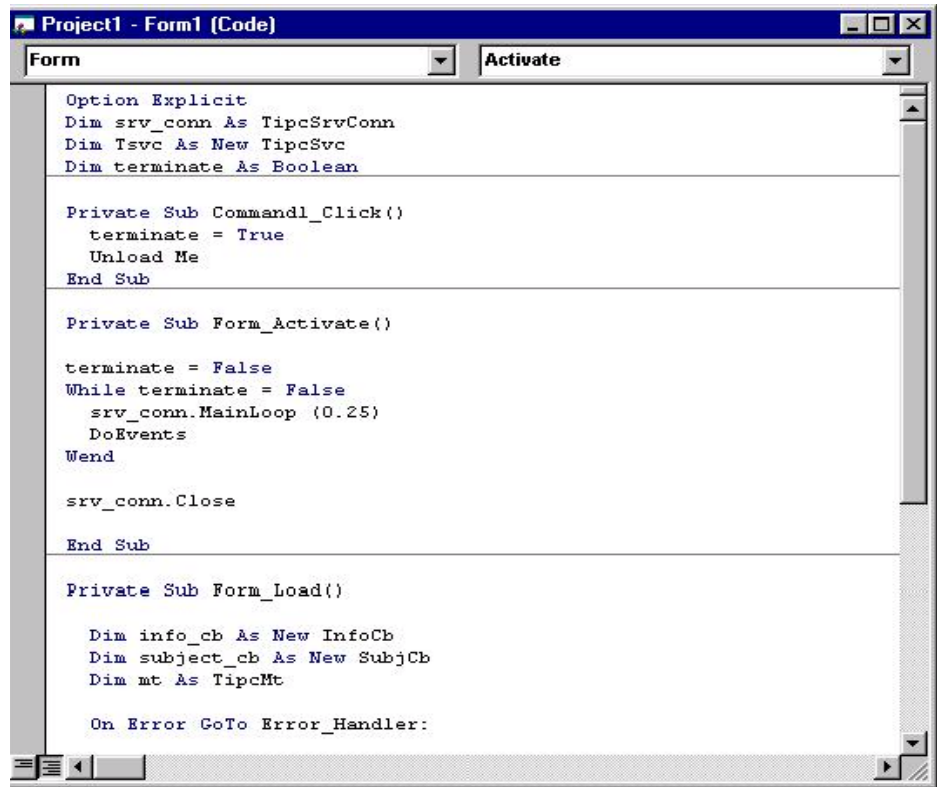
```

terminate = True
Unload Me

```

Figure 20 is an example of your program.

Figure 20 Example of Your Program



23. Close the code window.
24. From the General Controls palette, select the TextBox icon and draw a box in the Form window.
25. While the TextBox is selected in the Form window, in the Properties window change the Width value to 2555.
26. While both the Visual Basic and the SmartSockets Command Prompt windows are visible, start the Visual Basic program. Notice that there is now a Text box displayed above the Quit button in the Form window. Do not click the Quit button.

Example Programs

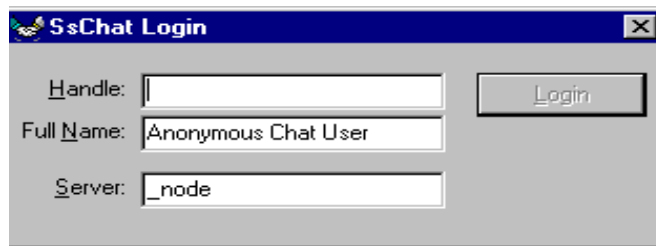
Now that you have written a simple SmartSockets program, this section presents several programs that use some of the more advanced SmartSockets features. These programs are not presented in a step-by-step instructional format, rather portions of the Visual Basic code for each program are reviewed and key elements are discussed.

SSChat: Multi-User Chat Room

The SSChat program is an RTclient that implements a simple real-time, multi-user chat room. The example files are located in the installation directory under `Examples\COM\sschat`. The SSChat program is written using the SmartSockets publish-subscribe technology to implement this function with a minimum amount of code. The entire SSChat program is under 200 lines of Visual Basic code.

The SSChat application starts with the Login window, as shown in Figure 21. The Login window is a standard Visual Basic form named `frmLogin`. It allows the user to input their full name and handle, and specify to which RTserver to connect.

Figure 21 SSChat Login Window



The essential details of SSChat's implementation are included within the main form, `frmMain`.

Exploring `frmMain`

There are several form-level global variables, found under (General) (Declarations), defined as:

```
Dim mtHi As TipcMt
Dim mtBye As TipcMt
Dim msgHi As TipcMsg
Dim msgOut As TipcMsg
Dim RTserver_Conn As TipcSrvConn
```

The first two variables, `mtHi` and `mtBye`, hold `TipcMt` objects specifying two of the custom message types `SSChat` defines. The second two, `msgHi` and `msgOut`, hold pre-built messages that are sent multiple times during an execution of `SSChat`. Finally, `RTserver_Conn` is a `TipcSrvConn` which manages the connection to `RTserver`.

Looking at the `Form_Load` method shows the actions taken upon entering the chat room. The custom message types are registered and the server name is configured. A connection to `RTserver` is created, process callbacks are instantiated using user callbacks implemented from `TipcProcessCb`, and a "Hi" (`mtHi`) message is constructed containing the user's name and chat handle, then sent.

```
Dim btn As Integer
Dim Tsvc As New TipcSvc
Dim server_names As String
Dim hi_cb As New HiCb
Dim bye_cb As New ByeCb
Dim data_cb As New DataCb

kScriptHeightOffset = Height - ctlScript.Height
kScriptMarginOffset = Width - ctlScript.RightMargin
kScriptWidthOffset = Width - ctlScript.Width
kMessageTopOffset = Height - ctlMessage.Top
kMessageWidthOffset = Width - ctlMessage.Width
ctlStatus.Panels(1) = fmLogin.ctlHandle
ctlStatus.Panels(2) = fmLogin.ctlName

ctlScript.SelColor = QBColor(1)
ctlScript.SelText = "[ " + str(Now) + " ]"
ctlScript.SelColor = QBColor(0)

On Error GoTo ErrorHandler

Set mtHi = Tsvc.MtCreate("hi", 1, "str str int2")
Set mtBye = Tsvc.MtCreate("bye", 2, "str")

Set msgOut = Tsvc.MsgCreate(Tsvc.MtLookup("string_data"))
msgOut.Dest = "_all"
msgOut.Fields.Add fmLogin.ctlHandle.Text

Set msgHi = msgOut.Clone
msgHi.Type = mtHi

msgHi.Fields.Add fmLogin.ctlName.Text
msgHi.Fields.Add Int(True)

server_names = ""
If Len(fmLogin.ctlServer) > 0 Then
    server_names = fmLogin.ctlServer
End If
Set RTserver_Conn = Tsvc.TipcSrvConnCreate(fmLogin.ctlHandle,
"rtworks", server_names, "")

RTserver_Conn.Open
RTserver_Conn.SetStdSubjectSubscribe True

HiCb_parms.Add RTserver_Conn
HiCb_parms.Add msgHi
HiCb_parms.Add msgOut
HiCb_parms.Add ctlScript
RTserver_Conn.ProcessCbCreate hi_cb, mtHi, 1, HiCb_parms
RTserver_Conn.ProcessCbCreate bye_cb, mtBye, 1, ctlScript
```

```

RTserver_Conn.ProcessCbCreate data_cb, Tsvc.MtLookup("string_data"),
1, ctlScript

RTserver_Conn.MsgSend msgHi
RTserver_Conn.Flush

Exit Sub

ErrorHandler:

Unload Me
btn = MsgBox(Err.Description)

End Sub

```



By cloning the msgOut message, the Dest property is the same for msgHi. Appending True requests other SSChat clients to reply upon receipt of the message.

To receive messages, the SSChat application must maintain a message loop. This is done in Form_Activate. It is a simple loop that processes events and checks for messages without a timeout to provide the best possible GUI response. The loop is terminated by the QuitListening flag set in Form_Unload.

```

Private Sub Form_Activate()

QuitListening = False
While QuitListening = False
RTserver_Conn.MainLoop (0)
DoEvents
Wend

End Sub

```

Once the form has been loaded, SSChat is in its primary mode, waiting for keystrokes from the user or messages from RTserver. Examine what happens when a key is pressed. Note that when you press the Enter key, the current text is sent out to the other chat participants.

```

Private Sub ctlMessage_KeyDown(KeyCode As Integer, Shift _
As Integer)
Dim str As String

If (Shift And vbShiftMask) = 0 Then
If KeyCode = vbKeyReturn Then
msgOut.NumFields = 1
str = ctlMessage
If Left(str, 1) = Chr(13) Then
str = Right(str, Len(str) - 2)
End If

msgOut.Fields.Add str
RTserver_Conn.MsgSend msgOut
RTserver_Conn.Flush
ctlMessage.Text = ""

End If
End If

```

Notes:

- Setting the NumFields property to 1 eliminates all but the first field of the message.
- msgOut is already mostly constructed; all you have to do is add the chat text, send and flush to guarantee immediate delivery.

Now that you have reviewed the sending process of chat text from SSChat, take a look at the three message processing events that handle the receiving process. The first ctlHiCb's TpcMsg event receives the "Hi" messages:

Implements TpcProcessCb

```
Private Sub TpcProcessCb_TpcMsg(ByVal conn As Object, ByVal pMsg As
TIPCCOMLib.ITpcMsg, ByVal arg As Variant)

    Dim msgIn As TpcMsg
    Dim msgHi As TpcMsg
    Dim msgOut As TpcMsg
    Dim srv_conn As TpcSrvConn
    Dim cb_parms As Collection
    Dim ctlScript As RichTextBox

    Set msgIn = pMsg
    Set cb_parms = arg

    Set srv_conn = cb_parms.Item(1)
    Set msgHi = cb_parms.Item(2)
    Set msgOut = cb_parms.Item(3)
    Set ctlScript = cb_parms.Item(4)

    If msgIn.Fields.Item(0) <> msgOut.Fields.Item(0) Then
        If msgIn.Fields.Item(2) Then
            msgHi.Dest = msgIn.Sender
            msgHi.NumFields = msgHi.NumFields - 1
            msgHi.Fields.Add Int(False)
            srv_conn.MsgSend msgHi
            srv_conn.Flush
        End If
    End If

    ctlScript.SelStart = 2000000000
    ctlScript.SelRTF = "{\par\par}"
    ctlScript.SelColor = QBColor(1)
    ctlScript.SelText = "[ " + msgIn.Fields.Item(1) + " connected as " +
msgIn.Fields.Item(0) + " ]"
    ctlScript.SelColor = QBColor(0)
    ctlScript.SelStart = 2000000000

End Sub
```

Now look at the code that handles the "Bye" messages, sent when a user leaves the chat room, terminating the SSChat process. It is much the same as the "Hi" message event handler, except it does not send any replies.

Implements TpcProcessCb

```
Private Sub TpcProcessCb_TpcMsg(ByVal conn As Object, ByVal pMsg As
TIPCCOMLib.ITpcMsg, ByVal arg As Variant)

    Dim msgIn As TpcMsg
```

```

Dim ctlScript As RichTextBox

Set msgIn = pMsg
Set ctlScript = arg

ctlScript.SelStart = 2000000000
ctlScript.SelRTF = "{\par\par}"
ctlScript.SelColor = QBColor(1)
ctlScript.SelText = "[ " + msgIn.Fields.Item(0) + " disconnected ]"
ctlScript.SelColor = QBColor(0)
ctlScript.SelStart = 2000000000

```

End Sub

The event handler for chat data messages is shown below (similar to the other two message events, it updates the output window with the originating chat user's handle and chat text):

Implements TipcProcessCb

```

Private Sub TipcProcessCb_TipcMsg(ByVal conn As Object, ByVal pMsg As
TIPCCOMLib.ITipcMsg, ByVal arg As Variant)

```

```

    Dim msgIn As TipcMsg
    Dim ctlScript As RichTextBox

    Set msgIn = pMsg
    Set ctlScript = arg

    ctlScript.SelStart = 2000000000
    ctlScript.SelRTF = "{\par\par}"
    ctlScript.SelBold = True
    ctlScript.SelText = msgIn.Fields.Item(0) + ": "
    ctlScript.SelBold = False
    ctlScript.SelText = msgIn.Fields.Item(1)
    ctlScript.SelStart = 2000000000

```

End Sub

In the code above, `Fields(0)` is the chat handle and `Fields(1)` is the message. Note that the `TipcMsg's` `NextStr` method could also have been used to access these values.

Finally, `SSChat's` `Form_Unload` subroutine that sends the "Bye" message and disconnects from `RTserver` (if a connection has been established) is shown below:

```

Private Sub Form_Unload(Cancel As Integer)
    Dim msgBye As TipcMsg
    Dim Tsvc As New TipcSvc

    QuitListening = True

    If Not RTserver_Conn Is Nothing Then
        If RTserver_Conn.Status <> tipcSrvConnNone Then
            Set msgBye = Tsvc.MsgCreate(mtBye)
            msgBye.Dest = msgOut.Dest
            msgBye.Fields.Add msgOut.Fields.Item(0)
            RTserver_Conn.MsgSend msgBye
            RTserver_Conn.Flush
            RTserver_Conn.Close
        End If
    End If

    If Not mtBye Is Nothing Then

```

```

        mtBye.Destroy
    End If

    If Not mtHi Is Nothing Then
        mtHi.Destroy
    End If

End Sub

```

Data Flow

The data flow in the SSChat program is:

1. Chat users joining the room are announced to the group members by publishing a "Hi" message containing information about the new user.
2. Other SSChat RTclients receive the new user's announcement and reply directly and exclusively to the originating RTclient, by publishing a "Hi" message back.
3. Upon receiving a "Hi" or "Bye" message, an SSChat RTclient updates its output window with the status change indicating which other client entered or left the room.
4. As data messages are received, SSChat displays these messages to its chat window.

WhoWhere: Electronic Message Board

The WhoWhere program is a graphical RTclient, implementing an in/out message board useful for tracking employee whereabouts. The WhoWhere program is located in the installation directory under `Examples\COM\whowhere`. WhoWhere is an electronic counterpart to the sign in and out boards commonly used in corporate offices. WhoWhere uses SmartSockets publish-subscribe technology. Other SmartSockets features demonstrated by this program are:

- Custom message types
- Process and error events
- Messages within messages
- Hierarchical naming scheme for publish-subscribe messages
- Publishing to entire subjects as well as individual RTclients

WhoWhere Overview

WhoWhere tracks employee whereabouts. Each employee belongs to a department. The department an employee belongs to specifies the SmartSockets subject used to set apart message board groups. This allows potentially thousands of users in hundreds of departments to be present on the same LAN or WAN, without causing conflict.

The WhoWhere application is composed of several Visual Basic forms and one code module. Most of the forms (those for logging in, managing the configuration and specifying leave information) are handled with the usual Visual Basic techniques; examination of the source code should be fairly self-explanatory. The important parts of the application are handled by the Display Board form and the modGlobals module.

The modGlobals Module

First, look at some of the data structures used in the modGlobals module:

```
Public Type configType
    Name As String
    Password As String
    Lunch As Integer
    Email As String
    Homepage As String
    MailApp As String
    WebApp As String
    Server As String
    Alerts As Integer
    Department As String
    EmailBrowser As Integer
End Type
Public Config As configType
```

The configType structure holds the local configuration information. This data is stored in the Windows registry by the SaveSettings subroutine and reloaded with LoadSettings. This allows the configuration to be persistent between executions of the application. The data includes the user's name, password and other configuration information. A subset of this information is maintained for all the other known employees on the message board as well, as shown in the userType data structure:

```
Public Type userType
    Who As String
    Email As String
    Homepage As String
    Message As String
    Where As String
    ReturnInfo As String
End Type
Public Myself As userType
Public Users(maxUsers) As userType
Public nUsers As Long
Public wwSubject As String
Public RTserver As TipcSrvConn
Public mtAnnounce As TipcMt
Public mtUpdate As TipcMt
```



```
Public mtResponse As TipcMt
Public Enum wwMessageTypes
wwAnnounce = 100
wwUpdate
wwResponse
End Enum
```

```
Public AnnounceCbParms As New Collection
```

Myself holds a copy of this user's message board information; Users() is the array that holds the actual message board information. nUsers is the number of users in the array and therefore the number of users displayed on the board. wwSubject holds the publish-subscribe subject name used for the WhoWhere client communication.

A global handle to the RTserver connection in the application is held in the RTserver variable. Additionally, WhoWhere defines three new message types identified by the numbers 100, 101, and 102 as the enumeration wwMessageTypes specifies; mtAnnounce, mtUpdate and mtResponse act as global references to these message type objects that are created.

Next, the initializeClient subroutine connects to RTserver and installs user-defined callbacks.

```
Public Sub initializeClient()

    Dim RTc As New TipcSvc
    Dim announce_an_cb As New TipcAnnounce
    Dim announce_resp_cb As New TipcAnnounce
    Dim update_cb As New TipcUpdateCb
    Dim default_cb As New TipcDefault
    Dim error_cb As New TipcErrorCb1

    If RTserver Is Nothing Then
        Set RTserver = RTc.TipcSrvConnCreate(Config.Name, programName,
        Config.server, "")
        RTserver.ServerNames = Config.server
        RTserver.project = programName
        RTserver.Open

        destroyMts

        ' create custom message types
        Set mtAnnounce = RTc.MtCreate("wwAnnounce", wwAnnounce, "int2
msg")
        Set mtResponse = RTc.MtCreate("wwResponse", wwResponse, "int2
msg")
        ' update is the msg that gets sent inside announce and response,
        ' and also by itself for updating status
        Set mtUpdate = RTc.MtCreate("wwUpdate", wwUpdate, "int2 str str
str str str str str")

        ' install event handlers for message types
        AnnounceCbParms.Add RTserver
        AnnounceCbParms.Add frmBoard
        RTserver.ProcessCbCreate announce_resp_cb, mtResponse, 1,
AnnounceCbParms
        RTserver.ProcessCbCreate announce_resp_cb, mtAnnounce, 1,
AnnounceCbParms
        RTserver.ProcessCbCreate update_cb, mtUpdate, 1, frmBoard
        RTserver.DefaultCbCreate default_cb, 1, Nothing
        RTserver.ErrorCbCreate error_cb, 1, False
```

```

        ' set up the WhoWhere pub/sub subject and subscribe
    End If

    wwSubject = "/" + programName + "/" + Config.Department
    RTserver.SetStdSubjectSubscribe True
    Dim status As Boolean
    RTserver.GetSubscribe wwSubject, status
    If status = False Then
        RTserver.SubjectSetSubscribe wwSubject, True
    End If
    RTserver.Flush

    ' send a message announcing our arrival
    doUpdate

End Sub

```

Only message board traffic published to the wwSubject subject is seen by this client. It is this hierarchical naming feature of SmartSockets that accounts for its scalability. Without changing the client, and with only one level of partitioning (the department), a large number of separate message boards can coexist. This is demonstrated in the RTserver.SubjectSetSubscribe statement in the initializeClient() subroutine above.

As illustrated in the code, a call to the doUpdate subroutine is made. This sends the equivalent of a "Hello" message from this RTclient to the other users displaying this department's message board. The following code for doUpdate builds and sends an announcement message using the mtAnnounce object as a reference (notice how a second SmartSockets message of type mtUpdate is included inside the announcement message):

```

Public Sub doUpdate()

    ' build and send an announcement message
    ' for initially joining a message board subject
    ' or doing an 'update all'
    Dim RTc As New TipcSvc
    Dim am As TipcMsg, myData As TipcMsg
    Set myData = RTc.MsgCreate(mtUpdate)

    With myData
        .AppendInt2 messageFormat
        .AppendStr Myself.Who
        .AppendStr Myself.Email
        .AppendStr Myself.Homepage
        .AppendStr Myself.Where
        .AppendStr Myself.Message
        .AppendStr Myself.ReturnInfo
    End With

    Set am = RTc.MsgCreate(mtAnnounce)
    am.AppendInt2 messageFormat
    am.AppendMsg myData
    am.Dest = wwSubject
    RTserver.MsgSend am
    RTserver.Flush

End Sub

```

The following subroutine, `publishMyStatus`, is used to send an update message when the current user's status changes, for example, when they leave or return to the office. A message of type `mtUpdate` is sent alone this time, not included inside another message.

```
Private Sub publishMyStatus()

    ' build and send just an update message when
    ' our status changes (go to lunch, return, etc.)
    Dim RTc As New TipcSvc

    Dim myData As TipcMsg

    Set myData = RTc.MsgCreate(mtUpdate)
    With myData
        .AppendInt2 messageFormat
        .AppendStr Myself.Who
        .AppendStr Myself.Email
        .AppendStr Myself.Homepage
        .AppendStr Myself.Where
        .AppendStr Myself.Message
        .AppendStr Myself.ReturnInfo
        .Dest = wwSubject
    End With

    RTserver.MsgSend myData
    RTserver.Flush

End Sub
```

The `setAway` subroutine updates the buttons available on the WhoWhere graphical user interface (GUI). There are two subroutines, `goAway` and `comeBack`, that make calls to `publishMyStatus`, as shown here:

```
Public Sub goAway()
    publishMyStatus
    frmBoard.setAway True
End Sub

Public Sub comeBack()
    Myself.Where = inStatus
    publishMyStatus
    frmBoard.setAway False
End Sub
```

The Display Board Form

The Display Board Form is the main user-interface object of the WhoWhere application. This form displays the message board with dynamic updating and allows the user to interact with the program through the command buttons.

In the next example, the form's `Load` subroutine calls the other initialization routines such as `initializeGUI`, which positions and sizes various screen elements. `initializeClient` configures the environment for `SmartSockets` and subscribes to the relevant subjects.

The forms `Form_Activate` code runs a loop processing messages in the background. This is how the application receives and processes messages using the `TipcSrvConn::Mainloop` interface. Note that the loop must be terminated by setting a flag in the `Unload` subroutine.

The form's `Unload` code saves the program settings and disconnects from the `RTserver`.

```
Private Sub Form_Load()
    Me.Left = Val(GetSetting(programName, "WindowPos", "Left",
Str(Screen.Width - Me.Width)))
    Me.Top = Val(GetSetting(programName, "WindowPos", "Top", "0"))
    Me.Caption = programName + " v" + programversion
    initializeGUI
    noErrMsg = True
    initializeClient
    Me.Show
    Me.Visible = True
End Sub

Private Sub Form_Activate()

DoLoop = True
While DoLoop = True
    DoEvents
    RTserver.MainLoop (0)
Wend

End Sub

Private Sub Form_Unload(Cancel As Integer)
    DoLoop = False
    SaveSetting programName, "WindowPos", "Left", Str(Me.Left)
    SaveSetting programName, "WindowPos", "Top", Str(Me.Top)
    saveSettings
    RTserver.Close
    End
End Sub
```

As shown in the next example, the `comeBack` and `goAway` subroutines are called when the absence or return (`btnBack`) buttons are clicked:

```
Private Sub btnHome_Click()
    Myself.Where = "HOME"
    Myself.Message = "(left for the day)"
    Myself.ReturnInfo = "the next work day"
    goAway
    btnBack.SetFocus
End Sub

Private Sub btnLunch_Click()
    Myself.Where = "LUNCH"
    Myself.Message = "(at lunch)"
    Myself.ReturnInfo = "at " + Format(DateAdd("n", _
Config.Lunch, Time))
    goAway
    btnBack.SetFocus
End Sub

Private Sub btnExtended_Click()
    frmExtended.Show 1
    If extendedCancel Then Exit Sub
    goAway
```

```

    btnBack.SetFocus
End Sub

Private Sub btnBack_Click()
    comeBack
End Sub

```

The displayBoard subroutine is the most important in terms of user interface; it re-populates the message board with data from the Users() array. The board has nUsers+1 rows; the extra is used to display the column headings. The message board is implemented with a Microsoft MSFlexGrid ActiveX component:

```

Public Sub displayBoard()
    Dim i As Integer, wh As String

    board.Rows = nUsers + 1
    For i = 0 To nUsers - 1
        board.Row = i + 1
        board.Col = 0
        board.Text = Users(i).Who
        board.Col = 1

        wh = Users(i).Where
        If wh <> inStatus Then
            wh = wh + " - returns " + Users(i).ReturnInfo
        End If
        board.Text = wh
        board.CellAlignment = flexAlignLeftCenter
    Next i
    btnMail.Visible = False
    btnWeb.Visible = False
    board.Col = 1
    board.Row = 1
    board.ColSel = 1
    board.RowSel = board.Rows - 1
    board.Sort = flexSortStringNoCaseAsending

    If Config.Alerts Then
        Beep
    End If
End Sub

```

The TipcAnnounce callback class decodes announce and response messages, and dispatches the newUser subroutine to add a user to the message board. Note that it was instantiated and registered in Form_Load(). As shown this example, if TipcAnnounce processes an announcement message, it responds by publishing a response message directly back to the originator:

```

Implements TipcProcessCb

Private Sub TipcProcessCb_TipcMsg(ByVal conn As Object, ByVal pMsg As
TIPCComLib.ITipcMsg, ByVal arg As Variant)
    Dim m As TipcMsg
    Dim cb_parms As Collection
    Dim mver As Integer
    Dim m2 As TipcMsg
    Dim resp As TipcMsg
    Dim myData As TipcMsg
    Dim frm_board As frmBoard
    Dim SrvConn As TipcSrvConn
    Dim wwAnnounce As Integer
    Dim mtResponse As Integer
    Dim mtUpdate As Integer

```

```

Set cb_parms = arg
Set SrvConn = cb_parms.Item(1)
Set frm_board = cb_parms.Item(2)

Set m = pMsg
mver = m.NextInt2
If mver > messageFormat Then
    frm_board.wrongMessageFormat "ANNOUNCE", mver
Else
    Set m2 = m.NextMsg
    mver = m2.NextInt2
    frm_board.newUser m2
    ' if announcement, send a response; don't respond to responses
    If m.Type.num = wwAnnounce Then
        Set resp = TipcSvc1.MsgCreate(mtResponse)
        resp.Dest = m.Sender
        resp.AppendInt2 msgversion

        Set myData = TipcSvc1.MsgCreate(mtUpdate)
        With myData
            .AppendInt2 messageFormat
            .AppendStr Myself.Who
            .AppendStr Myself.Email
            .AppendStr Myself.Homepage
            .AppendStr Myself.Where
            .AppendStr Myself.Message
            .AppendStr Myself.ReturnInfo
        End With

        resp.AppendMsg myData
        SrvConn.MsgSend resp
        SrvConn.Flush
    End If
End If
End Sub

```

The TipcDefault callback handles unexpected messages received by the WhoWhere RTclient. Once again, this was instantiated and registered in the Form_Load subroutine. In this example program, a message box appears, notifying the user of the alert:

Implements TipcProcessCb

```

Private Sub TipcProcessCb_TipcMsg(ByVal conn As Object, ByVal pMsg As
TIPCCOMLib.ITipcMsg, ByVal arg As Variant)

```

```

    Dim m As TipcMsg
    Set m = pMsg
    MsgBox "Received unexpected message of type " + m.Type.Name
End Sub

```

The TipcErrorCb callback's TipcError event is invoked when a SmartSockets error occurs. In this case, the relevant error information is displayed to the user to acknowledge:

Implements TipcErrorCb

```
Private Sub TipcErrorCb_TipcError(ByVal c_err_num As Long, ByVal
os_err_num As Long, ByVal socket_err_num As Long, ByVal
smart_sockets_err_num As Long, ByVal arg As Variant)
```

```
Dim noErrMsg As Boolean
noErrMsg = arg
```

```
    If noErrMsg Then
        noErrMsg = False
        MsgBox "There has been a SmartSockets error!" + Chr(10) + _
            "C Error: " + Str(c_err_num) + Chr(10) + _
            "OS Error: " + Str(os_err_num) + Chr(10) + _
            "Socket Error: " + Str(socket_err_num), _
            vbExclamation + vbApplicationModal + vbOKOnly, "SmartSockets
Error"
        noErrMsg = True
    End If
End Sub
```

The newUser subroutine takes an update message as a parameter, and adds the user information contained within to the Users() array, first removing any old instance of the user. In this example, the number of users is incremented and displayBoard is called to refresh the form display:

```
Public Sub newUser(um As TipcMsg)
    Dim thisname As String
    thisname = um.NextStr

    ' if user already on board, remove them
    Dim i As Integer, wasRemoved As Boolean
    i = 0
    While (i < nUsers And Not wasRemoved)
        If Users(i).Who = thisname Then
            For j = i To nUsers - 2
                Users(j) = Users(j + 1)
            Next j
            nUsers = nUsers - 1
            wasRemoved = True
        End If
        i = i + 1
    Wend

    With Users(nUsers)
        .Who = thisname
        .Email = um.NextStr
        .Homepage = um.NextStr
        .Where = um.NextStr
        .Message = um.NextStr
        .ReturnInfo = um.NextStr
    End With

    nUsers = nUsers + 1
    displayBoard
    showCount
End Sub
```

As shown in the next example, the `updateUser` subroutine takes an update message as a parameter, and updates the user information contained within to the `Users()` array. If the user is not currently in the array, they are added. Like the `newUser` subroutine, it calls `displayBoard` to refresh the form display.

```
Public Sub updateUser(um As TipcMsg)
    Dim thisname As String, uIndex As Integer
    thisname = um.NextStr
    ' user already on board?
    uIndex = -1
    For i = 0 To nUsers - 1
        If Users(i).Who = thisname Then
            uIndex = i
            Exit For
        End If
    Next i

    ' add this user if necessary
    If uIndex = -1 Then
        uIndex = nUsers
        nUsers = nUsers + 1
    End If

    With Users(uIndex)
        .Who = thisname
        .Email = um.NextStr
        .Homepage = um.NextStr
        .Where = um.NextStr
        .Message = um.NextStr
        .ReturnInfo = um.NextStr
    End With

    displayBoard
    showCount
End Sub
```

The `btnUpdate` subroutine, as shown below, is invoked when the Update All button is clicked. It resets the user count and calls `doUpdate`, re-issuing an announce message. The other users' message board applications will see this and send response messages directly to the running `WhoWhere` RTclient, populating the `Users()` array as the message callback events are invoked.

```
Private Sub btnUpdate_Click()
    nUsers = 0
    doUpdate
    board.SetFocus
End Sub
```


Data Flow

The flow of messages used in the WhoWhere program is:

1. Users just joining the message board group (specified by the department configuration field) are announced to the group members by publishing an `mtAnnounce` message containing information specific to the new user.
2. WhoWhere RTclients receive the new user's announcement and reply directly and exclusively to the originating RTclient by publishing an `mtResponse` message containing their information.
3. Upon receiving an `mtResponse` message, a WhoWhere RTclient adds the enclosed user information to their message board.
4. All WhoWhere RTclients receive `mtUpdate` messages, processing and changing the updated information to their message board in real-time.
5. If you click Update All, the board information is discarded and re-built by starting over as if a new client subscribed to the message board.

Notes

There are some points to note when you examine all the WhoWhere source code:

- Although not shown here, the `frmConfiguration` form calls `initializeClient` upon successful completion (that is, when you click OK). This ensures that any changes made to the configuration are accurately reflected by the WhoWhere application's state.
- When WhoWhere is invoked for the first time, the Configuration dialog box appears where you set up your message board environment. Once you enter the information, the Log In window appears.
- There is message format version-control in the WhoWhere program. The first field in every message sent is added with a line similar to:
`message.AppendInt2 messageFormat`

This value is decoded with passages similar to the next example in the message processing events:

```
Dim mver As Integer
mver = message.NextInt2
If mver > messageFormat Then
    wrongMessageFormat "MESSAGETYPE", mver
Else
    ' process the message
End If
```

The code extracts the first field from the messages, a two-byte integer, and checks it against the global constant `messageFormat` (see `modGlobals` for the definition of `messageFormat`). This ensures that if newer versions of `WhoWhere`, with different message grammars are present in the same department, the older `RTclients` do not corrupt their data with incompatible messages. Your applications may need more sophisticated message version control. This can be implemented with the `UserProp` property of `TipcMsg` objects.

- Remember, placing break points in the source code and stepping through the code with `Step Into` (the F8 key) is the way to see the sequence of events being encountered by the program.
- The global constant `Debugging` (see `modGlobals`) can be set to a non-zero value. This enables printing of diagnostic information in the Visual Basic Immediate window, which is useful for grasping the overall operation of the `WhoWhere` `RTclient`.
- When you enter your name in the `Name` field of the Configuration window, enter your first name, followed by your last name (for example, Jane Smith). Names are automatically displayed on the message board in alphabetical order: last name appearing first, followed by a comma and then the first name (for example, Smith, Jane).

Chapter 4 **ActiveX Controls Reference**

TIBCO SmartSockets components provide several classes, or interfaces, each with a large set of methods (callable functions), properties (settings) and events (callbacks). This chapter serves as a reference to these interfaces.

Topics

- *Relation to the TIBCO SmartSockets C API, page 72*
- *TIBCO SmartSockets ActiveX Classes, page 72*

Relation to the TIBCO SmartSockets C API

Both the ActiveX interface and the C API upon which the ActiveX interface is built, allow you to send and receive messages asynchronously across a heterogeneous network. The ActiveX classes offer a method counterpart for most of the SmartSockets C API calls.

Some of the tasks that can be performed using the ActiveX classes are:

- execute user-defined callback functions when certain operations occur
- send messages with subjects to multiple processes using the RTserver
- detect network failures and initiate actions to recover
- send messages using different IPC protocols over a heterogeneous network of computers
- monitor the application with poll and watch capabilities

TIBCO SmartSockets ActiveX Classes

ActiveX SmartSockets messaging applications (RTclients) connect with a client-server method, where an RTserver deals directly with your application. This method provides the greatest function and is the standard way to develop typical SmartSockets RTclient applications. The SmartSockets ActiveX controls provide full RTserver interaction using the TipcSvc and TipcSrv classes. The SmartSockets ActiveX classes and their usage are:

TipcErrorCb Class	provides error-handling events to RTclient applications.
TipcMon Class	provides monitoring for a SmartSockets application.
TipcMsg Class	is used for constructing, manipulating, and destroying SmartSockets messages.
TipcMsgFields Class	provides access to the distinct data fields of a SmartSockets message.
TipcMt Class	is used for constructing, manipulating, looking up and destroying SmartSockets message types.
TipcProcessCb Class	provides message-handling events to RTclient applications.

TipcSrv Class	provides the RTclient functions, such as sending messages to an RTserver and specifying subject subscription.
TipcSubjects Class	is used for managing the list of publish-subscribe subjects.
TipcSvc Class	provides connectivity to an RTserver and access to the RTclient connection options.
TipcXml Class	is a user class for managing XML objects.

TipcConn is also a SmartSockets ActiveX class, but is not used by ActiveX developers. It is for internal use only.

TipcErrorCb Class

TipcErrorCb — class for handling SmartSockets errors.

Description	The TipcErrorCb component contains the TipcErrorCb class, providing the TipcError event that invokes when a SmartSockets error condition arises.
Methods and Properties	None
Events	TipcError

TipcError Event

The TipcError event is triggered when a SmartSockets error occurs. Place any error-handling code related to SmartSockets in this event.

Syntax	Private Sub <i>object</i>_TipcError (<i>conn</i>, <i>c_err_num</i>, <i>os_err_num</i>, <i>socket_err_num</i>, <i>smart_sockets_err_num</i>) where: <i>object</i> is an object expression that evaluates to a TipcErrorCb object. <i>conn</i> is a TipcConn class that specifies the connection on which the error occurred. <i>c_err_num</i> is a long integer that specifies the C error code indicating how the last C function in the ActiveX control failed. <i>os_err_num</i> is a long integer that specifies the operating system error code indicating how the last operating system function called by the ActiveX control failed. <i>socket_err_num</i> is a long integer that specifies the socket error code indicating how the last socket operation called by the ActiveX control failed. <i>smart_sockets_err_num</i> is a long integer that specifies the SmartSockets error code indicating how the last SmartSockets operation failed.
Remarks	Typically, only one of the four values passed to the event is a valid error code. All of the parameters are passed to the event by value. You must register the TipcErrorCb event with the AddErrorCb method for the TipcError event to be invoked.

C API TutErrNumGetC, TutErrNumGetOs, TutErrNumGetSocket, TutErrNumGet; also similar to user-defined functions called after registration with TipcSrvErrorCbCreate

Example This is an example of a TipcError event handler:

```
Dim er As String, us As String, c As TipcConn
Set c = conn
us = c.UniqueSubject
If (c_err_num <> 0) Then
    er = "C Error: " + Format(c_err_num)
End If
If (os_err_num <> 0) Then
    er = er + "OS Error: " + Format(os_err_num)
End If
If (socket_err_num <> 0) Then
    er = er + "Socket Error: " + Format(socket_err_num)
End If
If (smart_sockets_err_num <> 0) Then
    er = er + "Smart Socket Error: " +
        Format(smart_sockets_err_num)
End If
MsgBox "SmartSockets error!" + Chr(10) + er,
    vbDefaultButton1, "SS Error!"
```

TipcMon Class

TipcMon — class for monitoring SmartSockets applications.

Description The TipcMon class is used to monitor SmartSockets applications. See the *TIBCO SmartSockets User's Guide* for more information on monitoring.

Methods

- ClientBufferGetWatch Method
 - ClientBufferSetWatch Method
 - ClientGeneralPoll Method
 - ClientMsgRecvSetWatch Method
 - ClientMsgSendSetWatch Method
 - ClientMsgTypePoll Method
 - ClientNamesPoll Method
 - ClientOptionPoll Method
 - ClientSubscribeGetWatch Method
 - ClientSubscribeSetWatch Method
 - ClientTimePoll Method
 - GetIdentStr Method
 - ProjectNamesPoll Method
 - ServerBufferPoll Method
 - ServerConnPoll Method
 - ServerGeneralPoll Method
 - ServerNamesGetWatch Method
 - ServerNamesSetWatch Method
 - ServerRoutePoll Method
 - SetIdentStr Method
 - SubjectNamesPoll Method
 - SubjectSubscribeGetWatch Method
 - SubjectSubscribeSetWatch Method
- ClientBufferPoll Method
 - ClientCbPoll Method
 - ClientMsgRecvGetWatch Method
 - ClientMsgSendGetWatch Method
 - ClientMsgTrafficPoll Method
 - ClientNamesGetWatch Method
 - ClientNamesSetWatch Method
 - ClientSubjectPoll Method
 - ClientSubscribePoll Method
 - ClientTimeGetWatch Method
 - ClientTimeSetWatch Method
 - ProjectNamesGetWatch Method
 - ProjectNamesSetWatch Method
 - ServerConnGetWatch Method
 - ServerConnSetWatch Method
 - ServerMsgTrafficPoll Method
 - ServerNamesPoll Method
 - ServerOptionPoll Method
 - ServerTimePoll Method
 - SubjectNamesGetWatch Method
 - SubjectNamesSetWatch Method
 - SubjectSubscribePoll Method

Properties and Events None

TipcMon Class Methods

ClientBufferGetWatch Method

The ClientBufferGetWatch method determines if this RTclient is watching message-related buffer information for other RTclients.

Syntax `object.ClientBufferGetWatch (client_name, watch_status)`
 where:
object is an object expression that evaluates to a TipcMon object.
client_name is a string that specifies the name of RTclient for determining watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch for all RTclients matching the MonitorScope property.
watch_status is a Boolean value representing the watch status:

- True means that watching is active
- False means there is no watching activity

Return Type Boolean

C API TipcMonClientBufferGetWatch

Example This example determines if this RTclient is watching all clients for message-related buffer information:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ClientBufferGetWatch("/...", status)
If result = True Then
  If status = False Then
    Debug.Print "Client Buffer not being watched"
  Else
    Debug.Print "Client Buffer being watched"
  End If
Else
  Debug.Print "ClientBufferGetWatch failed!"
End If
```

ClientBufferPoll Method

The ClientBufferPoll method polls once for RTclient message-related buffer information.

Syntax `object.ClientBufferPoll (client_name)`

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/" . . . to indicate a poll of all RTclients matching the MonitorScope property.

Return Type Boolean

C API TipcMonClientBufferPoll

Example This example polls for RTclient buffer information:

```
Dim mt As TipcMt
Dim msg As TipcMsg

'Poll all clients including self
TipcMon1.ClientBufferPoll "/"...
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_buffer_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
If msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_client_buffer_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    Debug.Print "RTclient name = " & msg.NextStr
    Debug.Print "msg queue count = " & msg.NextInt4
    Debug.Print "msg queue byte count = " & msg.NextInt4
    Debug.Print "read buffer count = " & msg.NextInt4
    Debug.Print "write buffer count = " & msg.NextInt4
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend
```

ClientBufferSetWatch Method

The ClientBufferSetWatch method starts or stops the watching process for an RTclient regarding message-related buffer information.

Syntax `object.ClientBufferSetWatch (client_name, watch_status)`

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of RTclient for setting the watch status. Wildcard names are allowed, or use "/"... to indicate a watch of all RTclients matching the MonitorScope property.

watch_status is a Boolean value that sets the watch status:

- True starts the watching process
- False stops the watching process

Return Type Boolean

C API TipcMonClientBufferSetWatch

Example This example starts watching for message-related buffer information for all RTclients. A callback is registered to handle the status message:

```
Dim result As Boolean
Dim mt As New TipcMt

mt.Lookup "mon_client_buffer_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ClientBufferSetWatch("/...", True)
TipcSvc1.Srv.Flush
If result = False Then
    Debug.Print "ClientBufferSetWatch failed!"
End If
```

ClientCbPoll Method

The ClientBufferPoll method polls once for RTclient callback information.

Syntax `object.ClientCbPoll (client_name)`

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

Return Type Boolean

C API TipcMonClientCbPoll

Example This example polls for RTclient callback information:

```
Dim mt As TipcMt
Dim msg As TipcMsg

'Poll all clients including self
TipcMon1.ClientCbPoll "/"...
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_cb_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
If msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If

Set mt = TipcSvc1.MtLookup("mon_client_cb_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    Debug.Print "RTclient name = " & msg.NextStr
    Debug.Print "num global read cb = " & msg.NextInt4
    Debug.Print "num global write cb = " & msg.NextInt4
    Debug.Print "num global process cb = " & msg.NextInt4
    Debug.Print "num global queue cb = " & msg.NextInt4
    Debug.Print "num default cb = " & msg.NextInt4
    Debug.Print "num error cb = " & msg.NextInt4
    Debug.Print "num server create cb = " & msg.NextInt4
    Debug.Print "num server destroy cb = " & msg.NextInt4
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend
```

ClientGeneralPoll Method

The ClientGeneralPoll method polls once for RTclient general information.

Syntax *object*.ClientGeneralPoll (*client_name*)

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

Return Type Boolean

C API TipcMonClientGeneralPoll

Example This example polls for RTclient general information:

```

Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim i_array(32) As Long
Dim s_array(32) As String

'Poll all clients including self
TipcMon1.ClientGeneralPoll "/..."
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_general_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
If msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If

Set mt = TipcSvc1.MtLookup("mon_client_general_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    Debug.Print "RTclient: " & msg.NextStr
    Debug.Print "Ident: " & msg.NextStr
    Debug.Print "Node: " & msg.NextStr
    Debug.Print "User: " & msg.NextStr
    Debug.Print "Pid: " & msg.NextInt4
    Debug.Print "Project: " & msg.NextStr
    Debug.Print "Server: " & msg.NextStr
    Debug.Print "Architecture: " & msg.NextStr
    Debug.Print "Current SBRK: " & msg.NextInt4
    Debug.Print "Delta SBRK: " & msg.NextInt4
    Debug.Print "Integer format: " & msg.NextInt2
    Debug.Print "Real format: " & msg.NextInt2
    Debug.Print "LCN: " & msg.NextStr

    Debug.Print "Counted licenses:"
    numItems = msg.NextInt4Array(i_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & i_array(i)
    Next i

    Debug.Print "Extra licenses:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    Debug.Print "Subjects:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

```

ClientMsgRecvGetWatch Method

The ClientMsgRecvGetWatch method determines if this RTclient is watching received messages in an RTclient.

Syntax `object.ClientMsgRecvGetWatch (client_name, msg_type, watch_status)`

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for determining watch status. Wildcard names are allowed, or use "/" . . . " to indicate a watch of all RTclients matching the MonitorScope property.

msg_type is a string that specifies the name of the message for getting watch status. Use "@" to indicate a watch of all message types.

watch_status is a Boolean value representing the watch status:

- True means that watching is active
- False means there is no watching activity

Return Type Boolean

C API TipcMonClientMsgRecvGetWatch

Example This example determines if this RTclient is watching all message types for the current RTclient:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ClientMsgRecvGetWatch
           (TipcSvc1.Unique_Subject,
            "@", status)

If result = True Then
    If status = False Then
        Debug.Print "Messages received not being watched"
    Else
        Debug.Print "Messages received being watched"
    End If
Else
    Debug.Print "ClientMsgRecvGetWatch failed!"
End If
```

ClientMsgRecvSetWatch Method

The ClientMsgRecvSetWatch method starts or stops watching for messages received for RTclient.

Syntax `object.ClientMsgRecvSetWatch (client_name, msg_type, watch_status)`
 where:
object is an object expression that evaluates to a TipcMon object.
client_name is a string that specifies the name of the RTclient for setting the watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTclients matching the MonitorScope property.
msg_type is a string that specifies the name of the message for setting the watch status. Use "@" to indicate a watch of all message types.
watch_status is a Boolean value that sets the watch status:

- True starts the watching process
- False stops the watching process

Return Type Boolean

C API TipcMonClientMsgRecvSetWatch

Example This example starts watching for messages received by the current client. A callback is registered to handle the status message:

```
Dim result As Boolean
Dim mt As New TipcMt

mt.Lookup "mon_client_msg_recv_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ClientMsgRecvSetWatch
        (TipcSvc1.Unique_Subject,
         "@", True)
TipcSvc1.Srv.Flush
If result = False Then
    Debug.Print "ClientMsgRecvSetWatch failed!"
End If
```

ClientMsgSendGetWatch Method

The ClientMsgSendGetWatch method determines if this RTclient is watching sent messages in an RTclient.

Syntax `object.ClientMsgSendGetWatch (client_name, msg_type, watch_status)`

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for determining watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTclients matching the MonitorScope property.

msg_type is a string that specifies the name of the message for retrieving watch status. Use "@" to indicate a watch of all message types.

watch_status is a Boolean value representing the watch status:

- True means that watching is active
- False mean there is no watching activity

Return Type Boolean

C API TipcMonClientMsgSendGetWatch

Example This example determines if sent messages are watched for the current RTclient:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ClientMsgSendGetWatch
        (TipcSvc1.Unique_Subject,
         "@", status)
If result = True Then
    If status = False Then
        Debug.Print "Messages sent not being watched"
    Else
        Debug.Print "Messages sent being watched"
    End If
Else
    Debug.Print "ClientMsgSendGetWatch failed!"
End If
```

ClientMsgSendSetWatch Method

The ClientMsgSendSetWatch method starts or stops the watching of messages sent by an RTclient.

Syntax *object*.ClientMsgSendSetWatch (*client_name*, *msg_type*, *watch_status*)

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for setting the watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTclients matching the MonitorScope property.

msg_type is a string that specifies the name of the message for setting the watch status. Use "@" to indicate a watch of all message types.

watch_status is a Boolean value that sets the watch status:

- True starts the watching process
- False stops the watching process

Return Type Boolean

C API TipcMonClientMsgSendSetWatch

Example This example starts watching for messages sent by the current client. A callback is registered to handle the status message:

```
Dim result As Boolean
Dim mt As New TipcMt

mt.Lookup "mon_client_msg_send_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ClientMsgSendSetWatch
        (TipcSvc1.Unique_Subject,
         "@", True)

TipcSvc1.Srv.Flush
If result = False Then
    Debug.Print "ClientMsgSendSetWatch failed!"
End If
```

ClientMsgTrafficPoll Method

The ClientMsgTrafficPoll method polls once for RTclient message traffic.

Syntax *object*.ClientMsgTrafficPoll (*client_name*)

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/" . . ." to indicate a poll of all RTclients matching the MonitorScope property.

Return Type Boolean

C API TipcMonClientMsgTrafficPoll

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example polls for RTclient message traffic information:

```
Dim mt As TipcMt
Dim msg As TipcMsg

'Poll all clients including self
TipcMon1.ClientMsgTrafficPoll "/"..."
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup
    ("mon_client_msg_traffic_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
If msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If

'Process the poll result message
Set mt = TipcSvc1.MtLookup
    ("mon_client_msg_traffic_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    Debug.Print "RTclient name = " & msg.NextStr
    Debug.Print "total msgs send= " & msg.NextInt4
    Debug.Print "total msgs recv= " & msg.NextInt4
    Debug.Print "total bytes send = " & msg.NextInt4
    Debug.Print "total bytes recv = " & msg.NextInt4
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend
```

ClientMsgTypePoll Method

The ClientMsgTypePoll method polls once for RTclient message type information.

Syntax `object.ClientMsgTypePoll (client_name, msg_type)`

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/" . . ." to indicate a poll of all RTclients matching the MonitorScope property.

msg_type is a string that specifies the name of message type to poll for information. Use "@" to indicate all message types.

Return Type Boolean

C API TipcMonClientMsgTypePoll

Example This example polls for RTclient message type information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean

'Poll all clients including self
TipcMon1.ClientMsgTypePoll TipcSvc1.Unique_Subject, "@"
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_msg_type_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
If msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If

Set mt = TipcSvc1.MtLookup
    ("mon_client_msg_type_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    On Error Goto EndOfMessage
    eom = False
    Debug.Print "RTclient:" & msg.NextStr
    While Not eom
        Debug.Print " msg_type: " & msg.NextStr
        Debug.Print " number: " & msg.NextInt4
        Debug.Print " grammar: " & msg.NextStr
        Debug.Print " priority_known: " & msg.NextInt4
        Debug.Print " priority: " & msg.NextInt2
        Debug.Print " delivery_mode: " & msg.NextInt4
        Debug.Print " user property: " & msg.NextInt4
        Debug.Print " num read cb: " & msg.NextInt4
        Debug.Print " num write cb: " & msg.NextInt4
        Debug.Print " num process cb: " & msg.NextInt4
        Debug.Print " num queue cb: " & msg.NextInt4
        Debug.Print " total msg send: " & msg.NextInt4
        Debug.Print " total msg rcv: " & msg.NextInt4
        Debug.Print " total byte send: " & msg.NextInt4
        Debug.Print " total byte rcv: " & msg.NextInt4
    Wend
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend
EndOfMessage:
    eom = True
Exit Sub
```

ClientNamesGetWatch Method

The ClientNamesGetWatch method determines if this RTclient is watching RTclient names.

Syntax *object*.ClientNamesGetWatch (*watch_status*)

where:

object is an object expression that evaluates to a TipcMon object.

watch_status is a Boolean value representing the watch status:

- True means that watching is active
- False means there is no watching activity

Return Type Boolean

C API TipcMonClientNamesGetWatch

Example This example determines if an RTclient is watching client names:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ClientNamesGetWatch(status)
If result = True Then
    If status = False Then
        Debug.Print "Client names not being watched"
    Else
        Debug.Print "Client names being watched"
    End If
Else
    Debug.Print "ClientNamesGetWatch failed!"
End If
```

ClientNamesPoll Method

The ClientNamesPoll method polls once for RTclient names.

Syntax *object*.ClientNamesPoll

where *object* is an object expression that evaluates to a TipcMon object.

Return Type Boolean

C API TipcMonClientNamesPoll

Example This example polls for RTclient names:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array(32) As String

'Poll all clients including self
TipcMon1.ClientNamesPoll
TipcSvc1.Srv.Flush

'Process the poll result message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_names_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
```

```

While msg.IsNull = False
    Debug.Print "Client names:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    Debug.Print "Client info strings:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    Debug.Print "server names:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

```

ClientNamesSetWatch Method

The ClientNamesSetWatch method starts or stops the watching of RTclient names.

Syntax `object.ClientNamesSetWatch (watch_status)`
 where:
object is an object expression that evaluates to a TipcMon object.
watch_status is a Boolean value that sets the watch status:

- True starts the watching process
- False stops the watching process

Return Type Boolean

C API TipcMonClientNamesSetWatch

Example This example starts watching for RTclient names. A callback is registered to handle the status message:

```

Dim result As Boolean
Dim mt As New TipcMt
mt.Lookup "mon_client_names_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ClientNamesSetWatch(True)
TipcSvc1.Srv.Flush
If result = False Then
    Debug.Print "ClientNamesSetWatch failed!"
End If

```

ClientOptionPoll Method

The ClientOptionPoll method polls once for RTclient options.

Syntax `object.ClientOptionPoll (client_name, option_name)`

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of RTclient for polling. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

option_name is a string that specifies the name of the option to poll for information. Use "@" to indicate a poll of all options.

Return Type Boolean

C API TipcMonClientOptionPoll

Example This example polls for RTclient option information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean
Dim numItems As Long
Dim i As Long
Dim s_array(32) As String

'Poll all clients including self
TipcMon1.ClientOptionPoll "/...", "@"
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_option_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
If msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_client_option_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    On Error Goto EndOfMessage
    eom = False
    Debug.Print "RTclient:" & msg.NextStr
    While Not eom
        Debug.Print " option: " & msg.NextStr
        Debug.Print " type: " & msg.NextInt2
        Debug.Print " value: " & msg.NextStr
        Debug.Print " required: " & msg.NextInt4
        Debug.Print " legal values:"
        numItems = msg.NextStrArray(s_array(0), 32)
        For i = 0 To numItems - 1
            Debug.Print " " & s_array(i)
        Next i
    While Not eom
        Debug.Print " " & s_array(i)
    End While
    eom = True
End While
```

```

        msg.Destroy
        Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
    Wend

EndOfMessage:
    eom = True
Exit Sub

```

ClientSubjectPoll Method

The ClientSubjectPoll method polls once for the statistics associated with the RTclient subscriptions.

Syntax *object.ClientSubjectPoll (client_name, subject)*

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

subject is a string that specifies the name of the subject to poll for information. Use "@" to indicate a poll of all subjects.

Return Type Boolean

C API TipcMonClientSubjectPoll

Example This example polls for RTclient subject information:

```

Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean

'Poll all clients including self
TipcMon1.ClientSubjectPoll "/"..., "@"
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_subject_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
If msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_client_subject_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    On Error Goto EndOfMessage
    eom = False
    Debug.Print "RTclient:" & msg.NextStr

```

```

While Not eom
    Debug.Print " Subject = " & msg.NextStr
    Debug.Print " total msgs send = " & msg.NextInt4
    Debug.Print " total msgs recv = " & msg.NextInt4
    Debug.Print " total bytes send = " & msg.NextInt4
    Debug.Print " total bytes recv = " & msg.NextInt4
Wend
msg.Destroy
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend
EndOfMessage:
eom = True
Exit Sub

```

ClientSubscribeGetWatch Method

The ClientSubscribeGetWatch method determines if this RTclient is watching the subjects to which an RTclient is subscribing.

Syntax *object.ClientSubscribeGetWatch (client_name, watch_status)*

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for determining watch status. Wildcard names are allowed, or use "/"... to indicate a watch of all RTclients matching the MonitorScope property.

watch_status is a Boolean value representing the watch status:

- True means that watching is active
- False means there is no watching activity

Return Type Boolean

C API TipcMonClientSubscribeGetWatch

Example This example determines if the RTclient is watching the subjects to which an RTclient is subscribing:

```

Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ClientSubscribeGetWatch("/client1", status)
If result = True Then
    If status = False Then
        Debug.Print "Subjects not being watched"
    Else
        Debug.Print "Subjects being watched"
    End If
Else
    Debug.Print "ClientSubscribeGetWatch failed!"
End If

```


ClientSubscribePoll Method

The ClientSubscribePoll method polls once for the subjects to which an RTclient is subscribing.

Syntax `object.ClientSubscribePoll (client_name)`

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient to poll for the subject subscription. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

Return Type Boolean

C API TipcMonClientSubscribePoll

Example This example polls all RTclients for the subjects to which they are subscribed:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array(32) As String

'Poll all clients including self
TipcMon1.ClientSubscribePoll "/"...
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_subscribe_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
If msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If

'Process the poll result message
Set mt = TipcSvc1.MtLookup
    ("mon_client_subscribe_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    On Error Goto EndOfMessage
    eom = False
    While Not eom
        Debug.Print " RTclient subjects:" & msg.NextStr
        numItems = msg.NextStrArray(s_array(0), 32)
        For i = 0 To numItems - 1
            Debug.Print " " & s_array(i)
        Next i
    Wend
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

EndOfMessage:
    eom = True
Exit Sub
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

ClientSubscribeSetWatch Method

The ClientSubscribeSetWatch method starts or stops the watching of subjects to which an RTclient is subscribing.

Syntax `object.ClientSubscribeSetWatch (client_name, watch_status)`
 where:
object is an object expression that evaluates to a TipcMon object.
client_name is a string that specifies the name of the RTclient for setting watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTclients matching the MonitorScope property.
watch_status is a Boolean value that sets the watch status:

- True starts the watching process
- False stops the watching process

Return Type Boolean

C API TipcMonClientSubscribeSetWatch

Example This example starts watching for subjects to which an RTclient is subscribing. A callback is registered to handle the status message:

```
Dim result As Boolean
Dim mt As New TipcMt

mt.Lookup "mon_client_subscribe_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ClientSubscribeSetWatch("/...", True)
TipcSvc1.Srv.Flush
If result = False Then
    Debug.Print "ClientSubscribeSetWatch failed!"
End If
```

ClientTimeGetWatch Method

The ClientTimeGetWatch method determines if an RTclient is watching time.

Syntax `object.ClientTimeGetWatch (client_name, watch_status)`
 where:
object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for determining watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTclients matching the MonitorScope property.

watch_status is a Boolean value representing the watch status:

- True means that watching is active
- False means there is no watching activity

Return Type Boolean

C API TipcMonClientTimeGetWatch

Example This example determines if this RTclient is watching time:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ClientTimeGetWatch("/client1", status)
If result = True Then
    If status = False Then
        Debug.Print "Time not being watched"
    Else
        Debug.Print "Time being watched"
    End If
Else
    Debug.Print "ClientTimeGetWatch failed!"
End If
```

ClientTimePoll Method

The ClientTimePoll method polls once for RTclient time information.

Syntax *object*.ClientTimePoll (*client_name*)

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient to poll for time information. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTclients matching the MonitorScope property.

Return Type Boolean

C API TipcMonClientTimePoll

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example polls for time information from each RTclient:

```
Dim mt As TipcMt
Dim msg As TipcMsg

'Poll all clients including self
TipcMon1.ClientTimePoll "/"..."
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_time_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
If msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_client_time_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    Debug.Print " RTclient: " & msg.NextStr
    Debug.Print " current time: " & msg.NextReal8
    Debug.Print " current time: " & msg.NextStr
    Debug.Print " wall time: " & msg.NextReal8
    Debug.Print " wall time: " & msg.NextStr
    Debug.Print " CPU time: " & msg.NextReal8
    Debug.Print " frame count: " & msg.NextInt4
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend
```

ClientTimeSetWatch Method

The ClientTimeSetWatch method starts or stops the watching of time information.

Syntax `object.ClientTimeSetWatch (client_name, watch_status)`

where:

object is an object expression that evaluates to a TipcMon object.

client_name is a string that specifies the name of the RTclient for setting watch status. Wildcard names are allowed, or use "/"..." to indicate a watch of all RTclients matching the MonitorScope property.

watch_status is a Boolean value that sets the watch status:

- True starts the watching process
- False stops the watching process

Return Type Boolean

C API TipcMonClientTimeSetWatch

Example This example starts watching for time information for all clients. A callback is registered to handle the status message:

```
Dim result As Boolean
Dim mt As New TipcMt

mt.Lookup "mon_client_time_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ClientTimeSetWatch("/...", True)
TipcSvc1.Srv.Flush
If result = False Then
    Debug.Print "ClientTimeSetWatch failed!"
End If
```

GetIdentStr Method

The GetIdentStr method retrieves the monitoring identification string of the process.

Syntax *object*.GetIdentStr (*ident_str*)

where:

object is an object expression that evaluates to a TipcMon object.

ident_str is the retrieved identification string.

Return Type Boolean

C API TipcMonGetIdentStr

Example This example retrieves the identification string of the process:

```
Dim result As Boolean
Dim ident As String

result = TipcMon1.GetIdentStr(ident)
If result = True Then
    Debug.Print "Identification string: " & ident
End If
```

ProjectNamesGetWatch Method

The ProjectNamesGetWatch method determines if this RTclient is watching project names.

Syntax *object*.ProjectNamesGetWatch (*watch_status*)

where:

object is an object expression that evaluates to a TipcMon object.

watch_status is a Boolean value representing the watch status:

- True means that watching is active
- False means there is no watching activity

Return Type Boolean

C API TipcMonProjectNamesGetWatch

Example This example determines if this RTclient is watching project names:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ProjectNamesGetWatch(status)
If result = True Then
    If status = False Then
        Debug.Print "Project names not being watched"
    Else
        Debug.Print "Project names being watched"
    End If
Else
    Debug.Print "ProjectNamesGetWatch failed!"
End If
```

ProjectNamesPoll Method

The ProjectNamesPoll method polls once for project names.

Syntax *object*.ProjectNamesPoll

where *object* is an object expression that evaluates to a TipcMon object.

Return Type Boolean

C API TipcMonProjectNamesPoll

Example This example polls for RTclient names:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array(32) As String

'Poll RTserver for the list of project names
TipcMon1.ProjectNamesPoll
TipcSvc1.Srv.Flush
```

```

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_project_names_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    Debug.Print "Project names:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

```

ProjectNamesSetWatch Method

The ProjectNamesSetWatch method starts or stops the watching of project names.

Syntax `object.ProjectNamesSetWatch (watch_status)`
 where:
object is an object expression that evaluates to a TipcMon object.
watch_status is a Boolean value that sets the watch status:

- True starts the watching process
- False stops the watching process

Return Type Boolean

C API TipcMonProjectNamesSetWatch

Example This example starts watching for project names. A callback is registered to handle the status message:

```

Dim result As Boolean
Dim mt As New TipcMt

mt.Lookup "mon_project_names_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ProjectNamesSetWatch(True)
TipcSvc1.Srv.Flush
If result = False Then
    Debug.Print "ProjectNamesSetWatch failed!"
End If

```

ServerBufferPoll Method

The ServerBufferPoll method polls once for RTserver message-related buffer information.

Syntax *object*.ServerBufferPoll (*server_name*, *client_name*)

where:

object is an object expression that evaluates to a TipcMon object.

server_name is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/" . . ." to indicate a poll of all RTservers matching the MonitorScope property.

client_name is a string that specifies the name of the RTclient, connected to the RTserver, to poll. Use "@" to indicate a poll of all subjects.

Return Type Boolean

C API TipcMonServerBufferPoll

Example This example polls for RTserver message-related buffer information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean

'Poll all RTservers
TipcMon1.ServerBufferPoll "/...", "@"
TipcSvc1.Srv.Flush

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_server_buffer_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    On Error Goto EndOfMessage
    eom = False
    Debug.Print "RTserver:" & msg.NextStr
    While Not eom
        Debug.Print " RTclient: " & msg.NextStr
        Debug.Print " Message Q count = " & msg.NextInt4
        Debug.Print " Message Q byte count = " & msg.NextInt4
        Debug.Print " read buffer count = " & msg.NextInt4
        Debug.Print " write buffer count = " & msg.NextInt4
    Wend
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend
EndOfMessage:
    eom = True
Exit Sub
```


ServerConnGetWatch Method

The ServerConnGetWatch method determines if this RTclient is watching the RTserver connections.

Syntax *object*.ServerConnGetWatch (*watch_status*)
where:
object is an object expression that evaluates to a TipcMon object.
watch_status is a Boolean value representing the watch status:

- True means that watching is active
- False means there is no watching activity

Return Type Boolean

C API TipcMonServerConnGetWatch

Example This example determines if this RTclient is watching RTserver connections:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ServerBufferGetWatch(status)
If result = True Then
  If status = False Then
    Debug.Print "Server connections not being watched"
  Else
    Debug.Print "Server connections being watched"
  End If
Else
  Debug.Print "ServerConnGetWatch failed!"
End If
```

ServerConnPoll Method

The ServerConnPoll method polls once for RTserver connection information.

Syntax *object*.ServerConnPoll
where *object* is an object expression that evaluates to a TipcMon object.

Return Type Boolean

C API TipcMonServerConnPoll

Example This example polls for RTserver connections:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array(32) As String
Dim i_array(32) As Long

'Poll all RTservers
TipcMon1.ServerConnPoll
TipcSvc1.Srv.Flush

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_server_conn_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    Debug.Print "server 1 names:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    Debug.Print "server 2 names:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    Debug.Print "connection names:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    Debug.Print "weights:"
    numItems = msg.NextInt4Array(i_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & i_array(i)
    Next i

    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend
```

ServerConnSetWatch Method

The ServerConnSetWatch method starts or stops the watching of RTserver connections.

Syntax *object*.ServerConnSetWatch (*watch_status*)

where:

object is an object expression that evaluates to a TipcMon object.

watch_status is a Boolean value that sets the watch status:

- True starts the watching process
- False stops the watching process

Return Type Boolean

C API TipcMonServerConnSetWatch

Example This example starts watching for RTserver connections. A callback is registered to handle the status message:

```
Dim result As Boolean
Dim mt As New TipcMt

mt.Lookup "mon_server_conn_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ServerConnSetWatch(True)
TipcSvc1.Srv.Flush
If result = False Then
    Debug.Print "ServerConnSetWatch failed!"
End If
```

ServerGeneralPoll Method

The ServerGeneralPoll method polls once for RTserver general information.

Syntax *object*.ServerGeneralPoll (*server_name*)

where:

object is an object expression that evaluates to a TipcMon object.

server_name is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/" to indicate a poll of all RTservers matching the MonitorScope property.

Return Type Boolean

C API TipcMonServerGeneralPoll

Example This example polls for RTserver general information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array(32) As String

'Poll all RTservers
```

```

TipcMon1.ServerGeneralPoll "/..."
TipcSvc1.Srv.Flush

'Process the result message
Set mt = TipcSvc1.MtLookup("mon_server_general_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    Debug.Print "RTserver: " & msg.NextStr
    Debug.Print "Ident: " & msg.NextStr
    Debug.Print "Node: " & msg.NextStr
    Debug.Print "User: " & msg.NextStr
    Debug.Print "Pid: " & msg.NextInt4
    Debug.Print "Architecture: " & msg.NextStr
    Debug.Print "Current SBRK: " & msg.NextInt4
    Debug.Print "Delta SBRK: " & msg.NextInt4
    Debug.Print "Integer format: " & msg.NextInt2
    Debug.Print "Real format: " & msg.NextInt2
    Debug.Print "cmd file name: " & msg.NextStr
    Debug.Print "no daemon flag: " & msg.NextStr
    Debug.Print "cmd mode flag: " & msg.NextStr

    Debug.Print "direct clients:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    Debug.Print "direct servers:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    Debug.Print "server subscribe:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    Debug.Print "client subscribe:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

```

ServerMsgTrafficPoll Method

The ServerMsgTrafficPoll method polls once for RTserver message traffic.

Syntax *object*.ServerMsgTrafficPoll (*server_name*)

where:

object is an object expression that evaluates to a TipcMon object.

server_name is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/"... to indicate a poll of all RTservers matching the MonitorScope property.

Return Type Boolean

C API TipcMonServerMsgTrafficPoll

Example This example polls for RTserver message traffic information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean

'Poll all servers
TipcMon1.ServerMsgTrafficPoll "/"...
TipcSvc1.Srv.Flush

'Process the poll result message
Set mt = TipcSvc1.MtLookup
("mon_server_msg_traffic_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    On Error Goto EndOfMessage
    eom = False
    Debug.Print "RTserver:" & msg.NextStr
    While Not eom
        Debug.Print " RTclient: " & msg.NextStr
        Debug.Print " total msgs send = " & msg.NextInt4
        Debug.Print " total msgs recv = " & msg.NextInt4
        Debug.Print " total bytes send = " & msg.NextInt4
        Debug.Print " total bytes recv = " & msg.NextInt4
    Wend
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

EndOfMessage:
eom = True
Exit Sub
```

ServerNamesGetWatch Method

The ServerNamesGetWatch method determines if this RTclient is watching RTserver names.

Syntax *object*.ServerNamesGetWatch (*watch_status*)

where:

object is an object expression that evaluates to a TipcMon object.

`watch_status` is a Boolean value representing the watch status:

- True means watching is active
- False means there is no watching activity

Return Type Boolean

C API TipcMonServerNamesGetWatch

Example This example determines if this RTclient is watching server names:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ServerNamesGetWatch(status)
If result = True Then
    If status = False Then
        Debug.Print "Server names not being watched"
    Else
        Debug.Print "Server names being watched"
    End If
Else
    Debug.Print "ServerNamesGetWatch failed!"
End If
```

ServerNamesPoll Method

The ServerNamesPoll method polls once for RTserver names.

Syntax `object.ServerNamesPoll`
where *object* is an object expression that evaluates to a TipcMon object.

Return Type Boolean

C API TipcMonServerNamesPoll

Example This example polls for RTserver names:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array(32) As String

'Poll all servers
TipcMon1.ServerNamesPoll
TipcSvc1.Srv.Flush

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_server_names_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
```

```

While msg.IsNull = False
    Debug.Print "RTserver names:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    Debug.Print "RTserver info strings:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i

    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

```

ServerNamesSetWatch Method

The ServerNamesSetWatch method starts or stops the watching of RTserver names.

Syntax *object*.ServerNamesSetWatch (*watch_status*)

where:

object is an object expression that evaluates to a TipcMon object.

watch_status is a Boolean value that sets the watch status:

- True starts the watching process
- False stops the watching process

Return Type Boolean

C API TipcMonServerNamesSetWatch

Example This example starts watching for RTserver names. A callback is registered to handle the status message:

```

Dim result As Boolean
Dim mt As New TipcMt

mt.Lookup "mon_server_names_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ServerNamesSetWatch(True)
TipcSvc1.Srv.Flush
If result = False Then
    Debug.Print "ServerNamesSetWatch failed!"
End If

```

ServerOptionPoll Method

The ServerOptionPoll method polls once for RTserver options.

Syntax *object*.ServerOptionPoll (*server_name*, *option_name*)

where:

object is an object expression that evaluates to a TipcMon object.

server_name is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/"... to indicate a poll of all RTservers matching the MonitorScope property.

option_name is a string that specifies the name of the option to poll for information. Use "@" to indicate a poll of all options.

Return Type Boolean

C API TipcMonServerOptionPoll

Example This example polls for RTserver option information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean
Dim numItems As Long
Dim i As Long
Dim s_array(32) As String

'Poll all servers
TipcMon1.ServerOptionPoll "/...", "@"
TipcSvc1.Srv.Flush

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_server_option_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    On Error Goto EndOfMessage
    eom = False
    Debug.Print "RTserver:" & msg.NextStr
    While Not eom
        Debug.Print " option: " & msg.NextStr
        Debug.Print " type: " & msg.NextInt2
        Debug.Print " value: " & msg.NextStr
        Debug.Print " required: " & msg.NextInt4
        Debug.Print " legal values:"
        numItems = msg.NextStrArray(s_array(0), 32)
        For i = 0 To numItems - 1
            Debug.Print " " & s_array(i)
        Next i
    Wend
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

EndOfMessage:
    eom = True
Exit Sub
```


ServerRoutePoll Method

The ServerRoutePoll method polls once for RTserver routing information.

Syntax `object.ServerRoutePoll (server_name, dest_server_name)`
 where:
object is an object expression that evaluates to a TipcMon object.
server_name is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/"... to indicate a poll of all RTservers matching the MonitorScope property.
dest_server_name is a string that specifies the name of the destination RTserver whose route should be polled. Use to indicate a poll of all options.

Return Type Boolean

C API TipcMonServerRoutePoll

Example This example polls for RTserver routing information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean
Dim numItems As Long
Dim i As Long
Dim s_array(32) As String

'Poll all servers
TipcMon1.ServerRoutePoll "/...", "@"
TipcSvc1.Srv.Flush

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_server_route_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
  On Error Goto EndOfMessage
  eom = False
  Debug.Print "RTserver:" & msg.NextStr
  While Not eom
    Debug.Print " destination RTserver: " & msg.NextStr
    Debug.Print " connected servers:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
      Debug.Print " " & s_array(i)
    Next i
    Debug.Print " distance: " & msg.NextInt4
    Debug.Print " route to server:"
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
      Debug.Print " " & s_array(i)
    Next i
  Wend
  msg.Destroy
  Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

EndOfMessage:
```

```
eom = True
Exit Sub
```

ServerTimePoll Method

The ServerTimePoll method polls once for RTserver time information.

Syntax *object*.ServerTimePoll (*server_name*)

where:

object is an object expression that evaluates to a TipcMon object.

server_name is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/" . . ." to indicate a poll of all RTservers matching the MonitorScope property.

Return Type Boolean

C API TipcMonServerTimePoll

Example This example polls for time information from each RTserver:

```
Dim mt As TipcMt
Dim msg As TipcMsg

'Poll all servers
TipcMon1.ServerTimePoll "/" . . ."
TipcSvc1.Srv.Flush

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_server_time_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    Debug.Print "RTserver: " & msg.NextStr
    Debug.Print " wall time: " & msg.NextReal8
    Debug.Print " wall time: " & msg.NextStr
    Debug.Print " CPU time: " & msg.NextReal8
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

SetIdentStr Method

The SetIdentStr method sets the monitoring identification string of a process.

Syntax *object*.SetIdentStr (*ident_str*)
 where:
 object is an object expression that evaluates to a TpcMon object.
 ident_str is the identification string to set.

Return Type Boolean

C API TpcMonSetIdentStr

Example This example sets the identification string of a process:

```
Dim result As Boolean

result = TpcMon1.SetIdentStr("Database driver")
If result = False Then
    Debug.Print "Unable to set identification string"
End If
```

SubjectNamesGetWatch Method

The SubjectNamesGetWatch method determines if this RTclient is watching subject names.

Syntax *object*.SubjectNamesGetWatch (*watch_status*)
 where:
 object is an object expression that evaluates to a TpcMon object.
 watch_status is a Boolean value representing the watch status:

- True means watching is active
- False means there is no watching activity

Return Type Boolean

C API TpcMonSubjectNamesGetWatch

Example This example determines if this RTclient is watching subject names:

```
Dim status As Boolean
Dim result As Boolean
```

```

result = TipcMon1.SubjectNamesGetWatch(status)
If result = True Then
    If status = False Then
        Debug.Print "Subject names not being watched"
    Else
        Debug.Print "Subject names being watched"
    End If
Else
    Debug.Print "SubjectNamesGetWatch failed!"
End If

```

SubjectNamesPoll Method

The SubjectNamesPoll method polls once for subject names.

Syntax *object*.SubjectNamesPoll

where *object* is an object expression that evaluates to a TipcMon object.

Return Type Boolean

C API TipcMonSubjectNamesPoll

Example This example polls for all subject names:

```

Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array(32) As String

'Poll RTserver for subject names
TipcMon1.SubjectNamesPoll
TipcSvc1.Srv.Flush

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_subject_names_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
    Debug.Print " Subject names:" & msg.NextStr
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
        Debug.Print " " & s_array(i)
    Next i
    msg.Destroy
    Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

```

SubjectNamesSetWatch Method

The SubjectNamesSetWatch method starts or stops the watching of subject names.

Syntax `object.SubjectNamesSetWatch (watch_status)`
 where:
object is an object expression that evaluates to a TpcMon object.
watch_status is a Boolean value that sets the watch status:

- True starts the watching process
- False stop the watching process

Return Type Boolean

C API TpcMonSubjectNamesSetWatch

Example This example starts watching for subjects names. A callback is registered to handle the status message:

```
Dim result As Boolean
Dim mt As New TpcMt

mt.Lookup "mon_subject_names_status"
TpcSvc1.Srv.AddProcessCb TpcProcessCb1.Object, mt
result = TpcMon1.SubjectNamesSetWatch(True)
TpcSvc1.Srv.Flush
If result = False Then
    Debug.Print "SubjectNamesSetWatch failed!"
End If
```

SubjectSubscribeGetWatch Method

The SubjectSubscribeGetWatch method determines if this RTclient is watching for RTclients that are subscribing to a subject.

Syntax `object.SubjectSubscribeGetWatch (subject_name, watch_status)`
 where:
object is an object expression that evaluates to a TpcMon object.
subject_name is a string that specifies the name of the subject for determining watch status. Wildcard names are allowed, or use "/. . ." to indicate a watch of all subjects matching the MonitorScope property.

watch_status is a Boolean value representing the watch status:

- True means that watching is active
- False means there is no watching activity

Return Type Boolean

C API TipcMonSubjectSubscribeGetWatch

Example This example determines if this RTclient is watching clients that are subscribed to a subject:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.SubjectSubscribeGetWatch("/new", status)
If result = True Then
    If status = False Then
        Debug.Print "/new subject not being watched"
    Else
        Debug.Print "/new subject being watched"
    End If
Else
    Debug.Print "SubjectSubscribeGetWatch failed!"
End If
```

SubjectSubscribePoll Method

The SubjectSubscribePoll method polls once for the clients that are subscribing to a subject.

Syntax *object*.SubjectSubscribePoll (*subject_name*)

where:

object is an object expression that evaluates to a TipcMon object.

subject_name is a string that specifies the name of the subject to poll for subject subscription information. Wildcard names are allowed, or use "/. . ." to indicate a poll of all subjects matching the MonitorScope property.

Return Type Boolean

C API TipcMonSubjectSubscribePoll

Example This example polls for all clients that are subscribing to "/db":

```
Dim mt As TipcMt, msg As TipcMsg, i As Long
Dim numItems As Long, eom As Boolean
Dim s_array(32) As String
```

```

'Poll "/db" subject for all clients
TipcMon1.SubjectSubscribePoll "/db"
TipcSvc1.Srv.Flush

'Process the poll result message
Set mt = TipcSvc1.MtLookup
      ("mon_subject_subscribe_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
  On Error Goto EndOfMessage
  eom = False
  While Not eom
    Debug.Print "Subject:" & msg.NextStr
    numItems = msg.NextStrArray(s_array(0), 32)
    For i = 0 To numItems - 1
      Debug.Print " " & s_array(i)
    Next i
  Wend
  msg.Destroy
  Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
Wend

EndOfMessage:
  eom = True

```

SubjectSubscribeSetWatch Method

The SubjectSubscribeSetWatch method starts or stops the watching of clients that are subscribing to a subject.

Syntax `object.SubjectSubscribeSetWatch (subject_name, watch_status)`

where:

- object* is an object expression that evaluates to a TipcMon object.
- client_name* is a string that specifies the name of the subject for setting watch status. Wildcard names are allowed, or use "/"... to indicate a watch of all subjects matching the MonitorScope property.
- watch_status* is a Boolean value that sets the watch status:
- True starts the watching process
 - False stops the watching process

Return Type Boolean

C API TipcMonSubjectSubscribeSetWatch

Example This example starts watching for clients that are subscribed to "/db". A callback is registered to handle the status message:

```
Dim result As Boolean
Dim mt As New TipcMt

mt.Lookup "mon_subject_subscribe_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.SubjectSubscribeSetWatch("/db", True)
TipcSvc1.Srv.Flush
If result = False Then
    Debug.Print "SubjectSubscribeSetWatch failed!"
End If
```


TipcMsg Class

TipcMsg — class for constructing, manipulating and destroying SmartSockets messages

Description The TipcMsg class is the largest interface and is used to create, build up, decompose and set options for SmartSockets messages. See the *TIBCO SmartSockets User's Guide* for more information on message composition in SmartSockets RTclient applications.

Methods

- Ack Method
- AddNamedChar Method
- AddNamedInt2Array Method
- AddNamedInt4Array Method
- AddNamedMsgArray Method
- AddNamedReal4Array Method
- AddNamedReal8Array Method
- AddNamedStrArray Method
- AppendBinary Method
- AppendInt2 Method
- AppendInt4 Method
- AppendMsg Method
- AppendReal4 Method
- AppendReal8 Method
- AppendStr Method
- AppendUnknown Method
- Create Method
- Destroy Method
- GetNamedChar Method
- GetNamedInt2Array Method
- GetNamedInt4Array Method
- GetNamedMsgArray Method
- GetNamedReal4Array Method
- GetNamedReal8Array Method
- AddNamedBinary Method
- AddNamedInt2 Method
- AddNamedInt4 Method
- AddNamedMsg Method
- AddNamedReal4 Method
- AddNamedReal8 Method
- AddNamedStr Method
- AddNamedXml Method
- AppendChar Method
- AppendInt2Array Method
- AppendInt4Array Method
- AppendMsgArray Method
- AppendReal4Array Method
- AppendReal8Array Method
- AppendStrArray Method
- Clone Method
- CurrentType Method
- GetNamedBinary Method
- GetNamedInt2 Method
- GetNamedInt4 Method
- GetNamedMsg Method
- GetNamedReal4 Method
- GetNamedReal8 Method
- GetNamedStr Method

- GetNamedStrArray Method
- IsNull Method
- NextChar Method
- NextInt2Array Method
- NextInt4Array Method
- NextMsgArray Method
- NextReal4Array Method
- NextReal8Array Method
- NextStrArray Method
- NextUnknown Method
- GetNamedXml Method
- NextBinary Method
- NextInt2 Method
- NextInt4 Method
- NextMsg Method
- NextReal4 Method
- NextReal8 Method
- NextStr Method
- NextType Method

Properties

- Current Property
- DeliveryMode Property
- Dest Property
- HeaderStrEncode Property
- NumFields Property
- Sender Property
- Type Property
- CurrentUnknown Property
- DeliveryTimeout Property
- Fields Property
- LbMode Property
- Priority Property
- SeqNum Property
- UserProp Property

Events None

TpcMsg Class Methods

Ack Method

The Ack method acknowledges the delivery of a message.

Syntax *object.Ack*

where *object* is an object expression that evaluates to a TpcMsg object.

Return Type None

C API TpcMsgAck

Example This example acknowledges a message:

```
Dim msg s TpcMsg

set msg = TpcSvc1.Srv.MsgNext(2.0)
if msg.isNull() = False Then
    msg.ack
End If
```

AddNamedBinary Method

The AddNamedBinary method appends a binary field to a message using a name.

Syntax *object.AddNamedBinary name, binary_data, binary_size*

where:

object is an object expression that evaluates to a TpcMsg object.

name is a string that specifies the field name to add to a message.

binary_data is the binary data to add to a message.

binary_size is a long integer that specifies the number of bytes in the binary data.

Return Type None

C API TpcMsgAddNamedBinary

Example This example adds a named binary field to a message:

```
Dim blob As string
Dim i As Long
For i =1 to 64
blob = blob + Chr(i Mod 255)
Next i
msg.AddNamedBinary "binary_data", blob, 64
```

AddNamedChar Method

The AddNamedChar method adds a named character field (field type CHAR, which is a one-byte integer) to a message. The actual ActiveX data type passed is a two-byte integer to better match the common Chr() and Asc() functions' data type.

Syntax *object.AddNamedChar name, char_data*
where:
object is an object expression that evaluates to a TipCMsg object.
name is a string that specifies the field name to add to a message.
char_data is the data to add to a message.

Return Type None

C API TipCMsgAddNamedChar

Example This example appends a character field to a message:

```
Dim ch As Integer
ch = Asc("C")
msg.AddNamedChar "char_data" ch
```

AddNamedInt2 Method

The AddNamedInt2 method adds a named two-byte integer field (field type INT2) to a message.

Syntax *object.AddNamedInt2 name, int2_data*
where:
object is an object expression that evaluates to a TipCMsg object.
name is a string that specifies the field name to add to a message.

int2_data is the two-byte integer data to add to a message.

Return Type	None
C API	TipcMsgAddNamedInt2
Example	This example appends a named two-byte integer field to a message: <pre>Dim i2 As Integer i2 =33 msg.AddNamedInt2 "int2_data", i2</pre>

AddNamedInt2Array Method

The AddNamedInt2Array method adds a named two-byte integer array field (field type INT2_ARRAY) to a message.

Syntax	<i>object</i>.AddNamedInt2Array <i>name</i>, <i>int2_array_data</i>, <i>array_size</i> where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>name</i> is a string that specifies the field name to add to a message. <i>int2_array_data</i> is an integer array that specifies the array of two-byte integers to add to a message. <i>array_size</i> is a long integer that specifies the number of elements in the array.
Return Type	None
C API	TipcMsgAddNamedInt2Array
Example	This example appends a named two-byte integer array field to a message: <pre>Dim i2_array(16) As Integer Dim i As Long for i =0 to 15 i2_array(i) = i Next i msg.AddNamedInt2Array "int2Array_data", i2_array(0), 16</pre>

AddNamedInt4 Method

The AddNamedInt4 method adds a named four-byte integer field (field type INT4) to a message.

Syntax *object*.AddNamedInt4 *name*, *int4_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
int4_data is a long integer that specifies the four-byte integer data to add to a message.

Return Type None

C API TipcMsgAddNamedInt4

Example This example appends a named four-byte integer field to a message:

```
Dim i4 As Long
i4 = 123456789
msg.AddNamedInt4 "Int4_data", i4
```

AddNamedInt4Array Method

The AddNamedInt4Array method adds a named four-byte integer array field (field type INT4_ARRAY) to a message.

Syntax *object*.AddNamedInt4Array *name*, *int4_array_data*, *array_size*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
int4_array_data is a long array that specifies the array of four-byte integers to add to a message.
array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API TipcMsgAddNamedInt4Array

Example This example appends a named four-byte integer array field to a message:

```
Dim i4_array(16) As Long
Dim i As Long
For i = 0 to 15
    i4_array(i) = i
Next i
msg.AddNamedInt4Array "Int4_Array_data", i4_array(0), 16
```

AddNamedMsg Method

The AddNamedMsg method adds a named message field (field type MSG) to a message.

Syntax *object.AddNamedMsg name, msg_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

msg_data is a TipcMsg class message string to add to the object message.

Return Type None

C API TipcMsgAddNamedMsg

Example This example appends a named message field to a message:

```
Dim msg2 As TipcMsg
Dim mt As TipcMt
Set mt = TipcSvc1.MtLookup("control")
Set msg2 = TipcSvc1.MsgCreate(mt)
msg2.AddNamedStr "quit force"
msg.AddNamedMsg "msg_data", msg2
```

AddNamedMsgArray Method

The AddNamedMsgArray method adds a named message array field (field type MSG_ARRAY) to a message.

Syntax *object.AddNamedMsgArray name, msg_array_data, array_size*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

msg_array_data is a *TipcMsgArray* class that specifies the *TipcMsg* objects to add to the object message.

array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API *TipcMsgAddNamedMsgArray*

Example This example appends a named message array field to a message:

```
Dim msg_array(4) As TipcMsg
Dim i As Long
Dim mt As TipcMt
Set mt = TipcSvc1.MtLookup("info")
For i =0 to 3
Set msg_array(i) = TipcSvc1.MsgCreate(mt)
msg_array(i).AppendStr "Msg Number: " & Format(i)
Next i
msg.AddNamedMsgArray "msg_array_data", msg_array(0), 4
```

AddNamedReal4 Method

The *AddNamedReal4* method adds a named four-byte real number field (field type REAL4) to a message.

Syntax *object.AddNamedReal4 name, real4_data*
 where:
object is an object expression that evaluates to a *TipcMsg* object.
name is a string that specifies the field name to add to a message.
real4_data is a single real number that specifies the four-byte real number to add to a message.

Return Type None

C API *TipcMsgAddNamedReal4*

Example This example appends a named four-byte real field to a message:

```
Dim r4 As Single
r4 = 123.45
msg.AddNamedReal4 "real4_data", r4
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

AddNamedReal4Array Method

The AddNamedReal4Array method adds a named four-byte real number array field (field type REAL4_ARRAY) to a message.

Syntax *object.AddNamedReal4Array name, real4_array_data, array_size*

where:

object is an object expression that evaluates to a TipCMsg object.

name is a string that specifies the field name to add to a message.

real4_array_data is a single array that specifies the array of four-byte real numbers to add to a message.

array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API TipCMsgAddNamedReal4Array

Example This example appends a four-byte real array field to a message:

```
Dim r4_array(16) As Single
Dim i As Long
For i = 0 to 15
    r4_array(i) = 1234.123 + i
Next i
msg.AddNamedReal4Array "real4_array_data", r4_array(0), 16
```

AddNamedReal8 Method

The AddNamedReal8 method adds a named eight-byte real number field (field type REAL8) to a message.

Syntax *object.AddNamedReal8 name, real8_data*

where:

object is an object expression that evaluates to a TipCMsg object.

name is a string that specifies the field name to add to a message.

real8_data is the eight-byte real number to add to a message.

Return Type None

C API TipCMsgAddNamedReal8

Example This example adds a named eight-byte real field to a message:

```
Dim r8 As Double
r8 = 1234567.123456
msg.AddNamedReal8 "real8_data", r8
```

AddNamedReal8Array Method

The AddNamedReal8Array method adds a named eight-byte real number array field (field type REAL8_ARRAY) to a message.

Syntax *object.AddNamedReal8Array name, real8_array_data, array_size*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
real8_array_data is a double array that specifies the array of eight-byte real numbers to add to a message.
array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API TipcMsgAddNamedReal8Array

Example This example appends an eight-byte real field to a message:

```
Dim r8_array(16) As Double
Dim i As Long
For i =0 to 15
r8_array(i) = 1234567.123456 + i
Next i
msg.AddNamedReal8Array "real8_array_data", r8_array(0), 16
```

AddNamedStr Method

The AddNamedStr method adds a named null-terminated character string field (field type STR) to a message. A string field is a special case of an array of characters.

Syntax *object.AddNamedStr name, str_data*
where:
object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

str_data is the string data to add to a message.

Return Type None

C API TipcMsgAddNamedStr

Example This example appends a string field to a message:

```
Dim s As String
s = "Test String")
msg.AddNamedStr "string_data", s
```

AddNamedStrArray Method

The AddNamedStrArray method adds a named string array field (field type STR_ARRAY) to the message.

Syntax *object.AddNamedStrArray name, str_array_data, array_size*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

str_array_data is a string array that specifies the array of string data to be added to a message.

array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API TipcMsgAddNamedStrArray

Example This example appends a string array field to a message:

```
Dim s_array(8) As String
Dim i As Long
For i = 0 to 7
s_array(i) = "Test String" & Format(i)
Next i
msg.AddNamedStrArray "string_array_data", s_array(0), 8
```

AddNamedXml Method

The AddNamedXml method adds a named TipcXml object (field type TIPCXML) to a message.

Syntax *object*.AddNamedXml *name*, *xml_object*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
xml_object is the TipcXml object to add to a message.

Return Type None

C API TipcMsgAddNamedXml

Example This example appends a named four-byte integer field to a message:

```
Dim tXml As New TipcXml
tXml.Create ("<data><data1><temp=hot/></data1></data>")
msg.AddNamedXml "Xml_data", tXml
```

AppendBinary Method

The AppendBinary method appends a binary field to a message. A binary field has the field type BINARY, which is a non-restrictive array of characters. Binary fields are useful for transferring images or large amounts of opaque data.

Syntax *object*.AppendBinary *binary_data*, *binary_size*
where:
object is an object expression that evaluates to a TipcMsg object.
binary_data is a string that specifies the binary data to append to a message.
binary_size is a long integer that specifies the number of bytes in the binary data.

Return Type None

C API TipcMsgAppendBinary

Example This example appends a binary field to a message:

```
Dim blob As string
Dim i As Long
```

```

For i = 1 to 64
    blob = blob + Chr(i Mod 255)
Next i
msg.AppendBinary blob, 64

```

AppendChar Method

The AppendChar method appends a character field (field type CHAR, which is a one-byte integer) to a message. The actual ActiveX data type passed is a two-byte integer to better match the common Chr() and Asc() functions' data type.

Syntax *object.AppendChar char_data*
 where:
object is an object expression that evaluates to a TipcMsg object.
char_data is a integer that specifies the one-byte of data to append to a message.

Return Type None

C API TipcMsgAppendChar

Example This example appends a character field to a message:

```

Dim ch As Integer
ch = Asc("C")
msg.AppendChar ch

```

AppendInt2 Method

The AppendInt2 method appends a two-byte integer field (field type INT2) to a message.

Syntax *object.AppendInt2 int2_data*
 where:
object is an object expression that evaluates to a TipcMsg object.
int2_data is a integer that specifies the two-byte integer data to append to a message.

Return Type None

C API TipcMsgAppendInt2

Example This example appends a two-byte integer field to a message:

```
Dim i2 As Integer
i2 = 33
msg.AppendInt2 i2
```

AppendInt2Array Method

The AppendInt2Array method appends a two-byte integer array field (field type INT2_ARRAY) to a message.

Syntax *object*.AppendInt2Array *int2_array_data*, *array_size*
where:
object is an object expression that evaluates to a TipcMsg object.
int2_array_data is an integer array that specifies the array of two-byte integers to append to a message.
array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API TipcMsgAppendInt2Array

Example This example appends a two-byte integer array field to a message:

```
Dim i2_array(16) As Integer
Dim i As Long
for i = 0 to 15
    i2_array(i) = i
Next i
msg.AppendInt2Array i2_array(0), 16
```

AppendInt4 Method

The AppendInt4 method appends a four-byte integer field (field type INT4) to a message.

Syntax *object*.AppendInt4 *int4_data*

where:

object is an object expression that evaluates to a TipcMsg object.

int4_data is a long integer that specifies the four-byte integer data to append to a message.

Return Type None

C API TipcMsgAppendInt4

Example This example appends a four-byte integer field to a message:

```
Dim i4 As Long
i4 = 123456789
msg.AppendInt4 i4
```

AppendInt4Array Method

The AppendInt4Array method appends a four-byte integer array field (field type INT4_ARRAY) to a message.

Syntax *object.AppendInt4Array int4_array_data, array_size*

where:

object is an object expression that evaluates to a TipcMsg object.

int4_array_data is an integer array that specifies the array of four-byte integers to append to a message.

array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API TipcMsgAppendInt4Array

Example This example appends a four-byte integer array field to a message:

```
Dim i4_array(16) As Long
Dim i As Long
For i = 0 to 15
    i4_array(i) = i
Next i
msg.AppendInt4Array i4_array(0), 16
```

AppendMsg Method

The AppendMsg method appends a message field (field type MSG) to a message.

Syntax *object.AppendMsg msg_data*
where:
object is an object expression that evaluates to a TipcMsg object.
msg_data is a TipcMsg class message string to append to the object message.

Return Type None

C API TipcMsgAppendMsg

Example This example appends a message field to a message:

```
Dim msg2 As TipcMsg
Dim mt As TipcMt

Set mt = TipcSvc1.MtLookup("control")
Set msg2 = TipcSvc1.MsgCreate(mt)
msg2.AppendStr "quit force"
msg.AppendMsg msg2
```

AppendMsgArray Method

The AppendMsgArray method appends a message array field (field type MSG_ARRAY) to a message.

Syntax *object.AppendMsgArray msg_array_data, array_size*
where:
object is an object expression that evaluates to a TipcMsg object.
msg_array_data is a TipcMsgArray class with the TipcMsg objects to append to the object message.
array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API TipcMsgAppendMsgArray

Example This example appends a message array field to a message:

```
Dim msg_array(4) As TipcMsg
Dim i As Long
Dim mt As TipcMt

Set mt = TipcSvc1.MtLookup("info")
For i = 0 to 3
    Set msg_array(i) = TipcSvc1.MsgCreate(mt)
    msg_array(i).AppendStr "Msg Number: " & Format(i)
Next i
msg.AppendMsgArray msg_array(0), 4
```

AppendReal4 Method

The AppendReal4 method appends a four-byte real number field (field type REAL4) to a message.

Syntax *object.AppendReal4 real4_data*

where:

object is an object expression that evaluates to a TipcMsg object.

real4_data is a single real number that specifies the four-byte real number to append to a message.

Return Type None

C API TipcMsgAppendReal4

Example This example appends a four-byte real field to a message:

```
Dim r4 As Single
r4 = 123.45
msg.AppendReal4 r4
```

AppendReal4Array Method

The AppendReal4Array method appends a four-byte real number array field (field type REAL4_ARRAY) to a message.

Syntax *object.AppendReal4Array real4_array_data, array_size*

where:

object is an object expression that evaluates to a TipcMsg object.

real4_array_data is a single array that specifies the array of four-byte real numbers to append to a message.

array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API TipcMsgAppendReal4Array

Example This example appends a four-byte real array field to a message:

```
Dim r4_array(16) As Single
Dim i As Long

For i = 0 to 15
    r4_array(i) = 1234.123 + i
Next i
msg.AppendReal4Array r4_array(0), 16
```

AppendReal8 Method

The AppendReal8 method appends an eight-byte real number field (field type REAL8) to a message.

Syntax *object.AppendReal8 real8_data*

 where:

object is an object expression that evaluates to a TipcMsg object.

real8_data is a double real number that specifies the eight-byte real number to append to a message.

Return Type None

C API TipcMsgAppendReal8

Example This example appends an eight-byte real field to a message:

```
Dim r8 As Double

r8 = 1234567.123456
msg.AppendReal8 r8
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

AppendReal8Array Method

The AppendReal8Array method appends an eight-byte real number array field (field type REAL8_ARRAY) to a message.

Syntax *object.AppendReal8Array real8_array_data, array_size*

where:

object is an object expression that evaluates to a TipcMsg object.

real8_array_data is a double array that specifies the array of eight-byte real numbers to append to a message.

array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API TipcMsgAppendReal8Array

Example This example appends an eight-byte real field to a message:

```
Dim r8_array(16) As Double
Dim i As Long

For i = 0 to 15
    r8_array(i) = 1234567.123456 + i
Next i
msg.AppendReal8Array r8_array(0), 16
```

AppendStr Method

The AppendStr method appends a null-terminated character string field (field type STR) to a message. A string field is a special case of an array of characters.

Syntax *object.AppendStr str_data*

where:

object is an object expression that evaluates to a TipcMsg object.

str_data is the string data to append to a message.

Return Type None

C API TipcMsgAppendStr

Example This example appends a string field to a message:

```
Dim s As String
s = "Test String")
msg.AppendStr s
```

AppendStrArray Method

The AppendStrArray method appends a string array field (field type STR_ARRAY) to a message.

Syntax *object.AppendStrArray str_array_data, array_size*
where:
object is an object expression that evaluates to a TipcMsg object.
str_array_data is a string array that specifies the array of string data to append to a message.
array_size is a long integer that specifies the number of elements in the array.

Return Type None

C API TipcMsgAppendStrArray

Example This example appends a string array field to a message:

```
Dim s_array(8) As String
Dim i As Long
For i = 0 to 7
    s_array(i) = "Test String" & Format(i)
Next i
msg.AppendStrArray s_array(0), 8
```

AppendUnknown Method

The AppendUnknown method appends a field with a value that is not known. In other words, the value is unset or n/a.

Syntax *object.AppendUnknown field_type*
where:
object is an object expression that evaluates to a TipcMsg object.
field_type is an integer that specifies the type of the new field.

Return Type	None
Remarks	Frequently, messages have a fixed number of fields in a constant order. In that case, omitting a field when data does not exist is not feasible. Setting the value of the field to unknown allows fields to be marked as having no data. This is especially valuable when the entire range of field values is valid within an application and there is need to denote that the field has no valid data.
C API	TipcMsgAppendUnknown
Example	This example appends a field of value unknown to a message: <pre>msg.AppendUnknown 7 'int4 type</pre>

Clone Method

The Clone method makes an identical copy of the TipcMsg message object.

Syntax	<pre>object.Clone</pre> where <i>object</i> is an object expression that evaluates to a TipcMsg object.
Return Type	TipcMsg
C API	TipcMsgClone
Example	This example makes an identical copy of the message object: <pre>Dim cmsg As New TipcMsg Set cmsg = msg.Clone</pre>

Create Method

The Create method initializes a TipcMsg object to be of the specified type. This method configures the internal settings of the object, making it available for appending data, sending messages, or other message-related operations.

Syntax	<pre>object.Create (mt)</pre> where: <i>object</i> is an object expression that evaluates to a TipcMsg object.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

mt is class `TipcMt`, and specifies the type of message to be created.

Return Type	<code>TipcMsg</code>
Remarks	See the <i>TIBCO SmartSockets User's Guide</i> for a list of the SmartSockets standard message types.
C API	<code>TipcMsgCreate</code>
Example	This example initializes a <code>TipcMsg</code> object: <pre>Dim msg as New TipcMsg msg.Create (TipcSvc1.MtLookup("numeric_data"))</pre>

CurrentType Method

The `CurrentType` method retrieves the current field's field type in a message.

Syntax	<code>object.CurrentType</code> where <i>object</i> is an object expression that evaluates to a <code>TipcMsg</code> object.
Return Type	<code>TipcFt</code>
C API	<code>TipcMsgGetCurrentFieldType</code>
Example	This example retrieves the field type of the current field in a message: <pre>Dim ft As TipcFt ft = msg.CurrentType</pre>

Destroy Method

The `Destroy` method destroys a message.

Syntax	<code>object.Destroy</code> where <i>object</i> is an object expression that evaluates to a <code>TipcMsg</code> object.
Return Type	None
Remarks	This method is not required, but is available for convenience.

C API TipcMsgDestroy

Example This example destroys a message:

```
Dim msg As TipcMsg

Set msg = TipcSvc1.Srv.MsgNext(2.0)
if msg.isNull() = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If
```

GetNamedBinary Method

The GetNamedBinary method retrieves the value of a binary field of a message using a name.

Syntax *object*.GetNamedBinary (*name*, *binary_data*)

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to retrieve from a message.

binary_data is a string that specifies the binary data to retrieve from a message.

Return Type Long

C API TipcMsgGetNamedBinary

Example This example retrieves the value of a named binary field in a message:

```
Dim blob As String
Dim iRetBytes As Integer
IRetBytes = msg.GetNamedBinary("binary_data", blob)
```

GetNamedChar Method

The GetNamedChar method retrieves the value of a character field in a message, using a name. If necessary, the field is converted to the data format used by the process.

Syntax *object*.GetNamedChar (*name*)

where:

object is an object expression that evaluates to a TipcMsg object.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

name is a string that specifies the field name to retrieve from a message.

Return Type Integer

C API TipcMsgGetNamedChar

Example This example retrieves the value of a named character field in the message:

```
Dim ch As Integer
ch = msg.GetNamedChar("char_data")
```

GetNamedInt2 Method

The GetNamedInt2 method retrieves the value of a named field in a message, which must be a two-byte integer field. If necessary, the field is converted to the data format used by the process.

Syntax *object*.GetNamedInt2 (*name*)
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.

Return Type None

C API TipcMsgGetNamedInt2

Example This example retrieves the value of a named two-byte integer field of the message:

```
Dim i2 As Integer
i2 = msg.GetNamedInt2("int2_data")
```

GetNamedInt2Array Method

The GetNamedInt2Array method retrieves the value of a named field in a message, which must be an array of two-byte integers. If necessary, the field is converted to the data format used by the process.

Syntax *object*.GetNamedInt2Array (*name*, *int2_array_data*, *array_size*)

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to retrieve from a message.

int2_array_data is an integer array used to hold the array of two-byte integers to retrieve from a message.

array_size is a long integer that specifies the maximum number of elements to place in *int2_array_data*.

Return Type	Long
Remarks	The return value indicates how many actual items were placed into <i>int2_array_data</i> . A negative return value means that there are more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into <i>int2_array_data</i> . However, no data is placed into <i>int2_array_data</i> when this occurs.
C API	TipcMsgGetNamedInt2Array
Example	This example retrieves the value of a named two-byte integer array field of the message: <pre>Dim numElements As Long Dim i2_array(12) As Integer numElements = msg.GetNamedInt2Array("i2_array_data", i2_array(0), 12) if numElements < 0 Then 'Error -- not enough space allocated End If</pre>

GetNamedInt4 Method

The GetNamedInt4 method retrieves the value of a named field in a message, which must be a four-byte integer field. If necessary, the field is converted to the data format used by the process.

Syntax	<i>object</i>.GetNamedInt4 (<i>name</i>) where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>name</i> is a string that specifies the field name of the field to retrieve from a message.
Return Type	Long

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

C API TipcMsgGetNamedInt4

Example This example retrieves the value of a named four-byte integer field of the message:

```
Dim i4 As Long
i4 = msg.GetNamedInt4("int4_data")
```

GetNamedInt4Array Method

The GetNamedInt4Array retrieves the value of a named field in a message, which must be an array of four-byte integers. If necessary, the field is converted to the data format used by the process.

Syntax *object*.GetNamedInt4Array (*name*, *int4_array_data*, *array_size*)
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.
int4_array_data is a long integer array used to hold the data retrieved from the message.
array_size is a long integer that specifies the maximum number of elements to place in *int4_array_data*.

Return Type Long

Remarks The return value indicates how many actual items were placed into *int4_array_data*. A negative return value means that there are more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into *int4_array_data*. However, no data is placed into *int4_array_data* when this occurs.

C API TipcMsgGetNamedInt4Array

Example This example retrieves the value of a named four-byte integer array field of the message:

```
Dim numElements As Long
Dim i4_array(12) As Long
numElements = msg.GetNamedInt4Array("int4_array_data", i4_array(0), 12)
if numElements < 0 Then
'Error -- not enough space allocated
End If
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

GetNamedMsg Method

The GetNamedMsg method retrieves the value of a named field in a message, which must be a message field.

Syntax `object.GetNamedMsg (name)`

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to retrieve from a message.

Return Type TipcMsg

C API TipcMsgGetNamedMsg

Example This example retrieves the value of a named message field of the message:

```
Dim msg2 As TipcMsg
msg2 = msg.GetNamedMsg("msg_data")
```

GetNamedMsgArray Method

The GetNamedMsgArray method retrieves the value of a named field in a message, which must be an array of messages.

Syntax `object.GetNamedMsgArray (name, msg_array_data, array_size)`

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to retrieve from a message.

msg_array_data is class TipcMsgArray, and is used to hold the data retrieved from the message.

array_size is a long integer that specifies the maximum number of elements to place in *msg_array_data*.

Return Type Long

Remarks The return value indicates how many actual items were placed into *msg_array_data*. A negative return value means that there are more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into *msg_array_data*. However, no data is placed into *msg_array_data* when this occurs.

C API TipcMsgGetNamedMsgArray

Example This example retrieves the value of a named message-array field of the message:

```
Dim numElements As Long
Dim msg_array(8) As TipcMsg
Dim msgName As String
Dim i As Long
numElements = msg.GetNamedMsgArray("msg_array_data", msg_array(0), 8)
if numElements < 0 Then
'Error -- not enough space allocated
Else
For i =0 to numElements - 1
msgName = msg_array(i).Type.Name
Next i
End if
```

GetNamedReal4 Method

The GetNamedReal4 message retrieves the value of a named field in a message, which must be a four-byte real number.

Syntax *object*.GetNamedReal4 (*name*)

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to retrieve from a message.

Return Type Single

C API TipcMsgGetNamedReal4

Example This example retrieves the value of a named four-byte real field of the message:

```
Dim r4 As Single
r4 = msg.GetNamedReal4("real4_data")
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

GetNamedReal4Array Method

The `GetNamedReal4Array` retrieves the value of a named field in a message, which must be an array of four-byte real numbers.

Syntax `object.GetNamedReal4Array (name, real4_array_data, array_size)`

where:

object is an object expression that evaluates to a `TipcMsg` object.

name is a string that specifies the field name to retrieve from a message.

real4_array_data is an array of single real numbers used to hold the data retrieved from the message.

array_size is a long integer that specifies the maximum number of elements to place in *real4_array_data*.

Return Type Long

Remarks The return value indicates how many actual items were placed into *real4_array_data*. A negative return value means that there are more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into *real4_array_data*. However, no data is placed into *real4_array_data* when this occurs.

C API `TipcMsgGetNamedReal4Array`

Example This example retrieves the value of a named four-byte real array field of the message:

```
Dim r4_array(8) As Single
Dim numElements As Long
numElements = msg.GetNamedReal4Array("real4_array_data", r4_array(0),
8)
if numElements < 0 Then
'Error -- not enough space allocated
End If
```

GetNamedReal8 Method

The `GetNamedReal8` method retrieves the value of a named field in a message, which must be an eight-byte real number. If necessary, the field is converted to the data format used by the process.

Syntax `object.GetNamedReal8 (name)`

where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.

Return Type Double
C API TipcMsgGetNamedReal8

Example This example retrieves the value of a named eight-byte real field of the message:

```
Dim r8 As Double  
r8 = msg.GetNamedReal8("real8_data")
```

GetNamedReal8Array Method

The GetNamedReal8Array method retrieves the value of a named field in a message, which must be an array of eight-byte real numbers.

Syntax *object*.GetNamedReal8Array (*name*, *real8_array_data*, *array_size*)
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.
real8_array_data is an array of double real numbers used to hold the data retrieved from the message.
array_size is a long integer that specifies the maximum number of elements to place in *real8_array_data*.

Return Type Long
Remarks The return value indicates how many actual items were placed into *real8_array_data*. A negative return value means that there are more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into *real8_array_data*. However, no data is placed into *real8_array_data* when this occurs.
C API TipcMsgGetNamedReal8Array

Example This example retrieves the value of a named eight-byte real array field of the message:

```
Dim r8_array(16) As Double  
Dim numElements As Long
```

```

numElements = msg.GetNamedReal8Array("r8_array_data", r8_array(0), 16)
if numElements < 0 Then
'Error -- not enough space allocated
End If

```

GetNamedStr Method

The GetNamedStr method retrieves the value of a named field, which must be a null-terminated character string field (field type STR). A string field is a special case of an array of characters. If necessary, the field is converted to the data format used by the process.

Syntax *object*.GetNamedStr (*name*)
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.

Return Type String

C API TipcMsgGetNamedStr

Example This example retrieves the value of a named string field of the message:

```

Dim s As String
s = msg.GetNamedStr("string_data")

```

GetNamedStrArray Method

The GetNamedStrArray method retrieves the value of a named field, which must be an array of null-terminated character string fields (field type STR). If necessary, the field is converted to the data format used by the process.

Syntax *object*.GetNamedStrArray (*name*, *str_array_data*, *array_size*)
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.
str_array_data is a string array used to hold the data retrieved from the message.
array_size is a long integer that specifies the maximum number of elements to place in *str_array_data*.

Return Type	Long
Remarks	The return value indicates how many actual items were placed into <i>str_array_data</i> . A negative return value means that there are more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into <i>str_array_data</i> . However, no data is placed into <i>str_array_data</i> when this occurs.
C API	TipcMsgGetNamedStrArray
Example	This example retrieves the value of a named string array field of the message: <pre>Dim s_array(8) As String Dim numElements As Long numElements = msg.GetNamedStrArray("my_str", s_array(0), 8) if numElements < 0 Then 'Error -- not enough space allocated End If</pre>

GetNamedXml Method

The GetNamedXml method retrieves the value of a named field in a message, which must be a TipcXml message field.

Syntax	<pre>object.GetNamedXml (name)</pre> where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>name</i> is a string that specifies the field name to retrieve from a message.
Return Type	TipcXml
C API	TipcMsgGetNamedXml
Example	This example retrieves the value of a named XML message field in the message: <pre>Dim tXml As New TipcXml Set tXml = msg.GetNamedXml("xml_data")</pre>

IsNull Method

The IsNull method evaluates the message to determine if it is null. Although a message object may exist (perhaps because of a `Dim msg As TipcMsg` statement), it might not be initialized to an actual SmartSockets message. The type of message might not be set or it might not have memory allocated to it. The IsNull method returns `True` if the message object is not yet a full SmartSockets message, `False` if it has been initialized.

Syntax `object.IsNull`

where *object* is an object expression that evaluates to a `TipcMsg` object.

Return Type Boolean, where:

`True` indicates the object is null and not yet a valid SmartSockets message. Do not send this message until further initialization has been performed.

`False` indicates the message is of a known type and has memory allocated to it. This message is not null and can be sent.

Remarks If `IsNull` returns `True`, use one of the `Lookup` or `Create` methods to initialize it. `TipcSrv`'s `MsgNext` method can return an `IsNull()=True` message.

C API No comparable C API function

Example This example evaluates a message to see if it is null:

```
Dim msg As New TipcMsg
if msg.IsNull = True
    msg.Create(TipcSvc1.MtLookup("control"))
End If
```

NextBinary Method

The `NextBinary` method retrieves the value of the current field of a message, which must be a binary field. If successful, it advances the current field to the next field in the message.

Syntax `object.NextBinary`

where *object* is an object expression that evaluates to a `TipcMsg` object.

Return Type String

C API `TipcMsgNextBinary`

Example This example retrieves the value of the current binary field of a message:

```
Dim blob As String
blob = msg.NextBinary
```

NextChar Method

The NextChar method retrieves the value of the current field of a message, which must be a character field. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextChar
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type Integer

C API TipcMsgNextChar

Example This example retrieves the value of the current character field of the message:

```
Dim ch As Integer
ch = msg.NextChar
```

NextInt2 Method

The NextInt2 method retrieves the value of the current field of a message, which must be a two-byte integer field. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextInt2
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type Integer

C API TipcMsgNextInt2

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example retrieves the value of the current two-byte integer field of the message:

```
Dim i2 As Integer
i2 = msg.NextInt2
```

NextInt2Array Method

The NextInt2Array method retrieves the value of the current field of a message, which must be an array of two-byte integers. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextInt2Array (*int2_array_data*,*array_size*)

where:

object is an object expression that evaluates to a TipcMsg object.

int2_array_data is an array of two-byte integers used to hold the data retrieved from the message.

array_size is a long integer that specifies the maximum number of elements to place in *int2_array_data*.

Return Type Long

Remarks The return value indicates how many actual items were placed into *int2_array_data*. A negative return value means that there were more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into *int2_array_data*. However, no data is placed into *int2_array_data* when this occurs.

C API TipcMsgNextInt2Array

Example This example retrieves the value of the current two-byte integer array field of the message:

```
Dim numElements As Long
Dim i2_array(12) As Integer

numElements = msg.NextInt2Array(i2_array(0), 12)
if numElements < 0 Then
    'Error -- not enough space allocated
End If
```

NextInt4 Method

The NextInt4 method retrieves the value of the current field of a message, which must be a four-byte integer field. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextInt4
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type Long

C API TipcMsgNextInt4

Example This example retrieves the value of the current four-byte integer field of the message:

```
Dim i4 As Long
i4 = msg.NextInt4
```

NextInt4Array Method

The NextInt4Array retrieves the value of the current field of a message, which must be an array of four-byte integers. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextInt4Array (*int4_array_data*, *array_size*)
where:
object is an object expression that evaluates to a TipcMsg object.
int4_array_data is an array of long integers used to hold the data retrieved from the message.
array_size is a long integer that specifies the maximum number of elements to place in *int4_array_data*.

Return Type Long

Remarks The return value indicates how many actual items were placed into *int4_array_data*. A negative return value means that there were more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into *int4_array_data*. However, no data is placed into *int4_array_data* when this occurs.

C API TipcMsgNextInt4Array

Example This example retrieves the value of the current four-byte integer array field of the message:

```
Dim numElements As Long
Dim i4_array(12) As Long

numElements = msg.NextInt4Array(i4_array(0), 12)
if numElements < 0 Then
    'Error -- not enough space allocated
End If
```

NextMsg Method

The NextMsg method retrieves the value of the current field of a message, which must be a message field. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextMsg

where *object* is an object expression that evaluates to a TipcMsg object.

Return Type TipcMsg

C API TipcMsgNextMsg

Example This example retrieves the value of the current message field of the message:

```
Dim msg2 As TipcMsg
Dim msgName As String
msg2 = msg.NextMsg
msgName = msg2.Type.Name
```

NextMsgArray Method

The NextMsgArray method retrieves the value of the current field of a message, which must be an array of messages. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextMsgArray (*msg_array_data*, *array_size*)
where:
object is an object expression that evaluates to a TipcMsg object.
msg_array_data is class TipcMsgArray, and is used to hold the data retrieved from the message.
array_size is a long integer that specifies the maximum number of elements to place in *msg_array_data*.

Return Type Long

Remarks The return value indicates how many actual items were placed into *msg_array_data*. A negative return value means that there were more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into *msg_array_data*. However, no data is placed into *msg_array_data* when this occurs.

C API TipcMsgNextMsgArray

Example This example retrieves the value of the current message array field of the message:

```
Dim numElements As Long
Dim msg_array(8) As TipcMsg
Dim msgName As String
Dim i As Long

numElements = msg.NextMsgArray(msg_array(0), 8)
if numElements < 0 Then
    'Error -- not enough space allocated
Else
    For i = 0 to numElements - 1
        msgName = msg_array(i).Type.Name
    Next i
End if
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

NextReal4 Method

The NextReal4 message retrieves the value of the current field of a message, which must be a four-byte real number. If successful, it advances the current field to the next field in the message.

Syntax	<i>object</i>.NextReal4 where <i>object</i> is an object expression that evaluates to a TipcMsg object.
Return Type	Single
C API	TipcMsgNextReal4
Example	This example retrieves the value of the current four-byte real field of the message: <pre>Dim r4 As Single r4 = msg.NextReal4</pre>

NextReal4Array Method

The NextReal4Array retrieves the value of the current field of a message, which must be an array of four-byte real numbers. If successful, it advances the current field to the next field in the message.

Syntax	<i>object</i>.NextReal4Array (<i>real4_array_data</i>, <i>array_size</i>) where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>real4_array_data</i> is an array of single real numbers used to hold the data retrieved from the message. <i>array_size</i> is a long integer that specifies the maximum number of elements to place in <i>real4_array_data</i>.
Return Type	Long
Remarks	The return value indicates how many actual items were placed into <i>real4_array_data</i>. A negative return value means that there were more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into <i>real4_array_data</i>. However, no data is placed into <i>real4_array_data</i> when this occurs.
C API	TipcMsgNextReal4Array

Example This example retrieves the value of the current four-byte real array field of the message:

```
Dim r4_array(8) As Single
Dim numElements As Long

numElements = msg.NextReal4Array(r4_array(0), 8)
if numElements < 0 Then
    'Error -- not enough space allocated
End If
```

NextReal8 Method

The NextReal8 method retrieves the value of the current field of a message, which must be an eight-byte real number. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextReal8
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type Double

C API TipcMsgNextReal8

Example This example retrieves the value of the current eight-byte real field of the message:

```
Dim r8 As Double
r8 = msg.NextReal8
```

NextReal8Array Method

The NextReal8Array method retrieves the value of the current field of a message, which must be an array of eight-byte real numbers. If successful, it advances the current field to the next field in the message.

Syntax *object*.NextReal8Array (*real8_array_data*, *array_size*)
where:
object is an object expression that evaluates to a TipcMsg object.

real8_array_data is an array of double real numbers used to hold the data retrieved from the message.

array_size is a long integer that specifies the maximum number of elements to place in *real8_array_data*.

Return Type	Long
Remarks	The return value indicates how many actual items were placed into <i>real8_array_data</i> . A negative return value means that there were more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into <i>real8_array_data</i> . However, no data is placed into <i>real8_array_data</i> when this occurs.
C API	TipcMsgNextReal8Array
Example	This example retrieves the value of the current eight-byte real array field of the message: <pre>Dim r8_array(16) As Double Dim numElements As Long numElements = msg.NextReal8Array(r8_array(0), 16) if numElements < 0 Then 'Error -- not enough space allocated End If</pre>

NextStr Method

The NextStr method retrieves the value of the current field, which must be a null-terminated character string field (field type STR). A string field is a special case of an array of characters. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax	<i>object</i> .NextStr where <i>object</i> is an object expression that evaluates to a TipcMsg object.
Return Type	String
C API	TipcMsgNextStr

Example This example retrieves the value of the current string field of the message:

```
Dim s As String
s = msg.NextStr
```

NextStrArray Method

The NextStrArray method retrieves the value of the current field, which must be an array of null-terminated character string fields (field type STR). If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax `object.NextStrArray (str_array_data, array_size)`
where:
object is an object expression that evaluates to a TipcMsg object.
str_array_data is a string array used to hold the data retrieved from the message.
array_size is a long integer that specifies the maximum number of elements to place in *str_array_data*.

Return Type Long

Remarks The return value indicates how many actual items were placed into *str_array_data*. A negative return value means that there were more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into *str_array_data*. However, no data is placed into *str_array_data* when this occurs.

C API TipcMsgNextStrArray

Example This example retrieves the value of the current string array field of the message:

```
Dim s_array(8) As String
Dim numElements As Long

numElements = msg.NextStrArray(s_array(0), 8)
if numElements < 0 Then
    'Error -- not enough space allocated
End If
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

NextType Method

The NextType method retrieves the field type of the next field in a message.

Syntax `object.NextType`

where *object* is an object expression that evaluates to a TpcMsg object.

Return Type TpcFt

C API TpcMsgNextType

Example This example retrieves the field type of the next field in a message:

```
Dim ft As TpcFt  
ft = msg.NextType
```

NextUnknown Method

The NextUnknown method retrieves the value of the current field, which must have an unknown value, as set by AppendUnknown.

Syntax `object.NextUnknown`

where *object* is an object expression that evaluates to a TpcMsg object.

Return Type None

C API TpcMsgNextUnknown

Example This example retrieves the value of the current character field of the message:

```
msg.nextUnknown
```

TipcMsg Class Properties

Current Property

The Current property sets or retrieves the current field of a message. This does not affect the number of fields in the message; new fields are still appended to the message.

Syntax *object*.Current[=*field_num*]
where:
object is an object expression that evaluates to a TipcMsg object.
field_num is a long integer that specifies the field number to use for the current field. Using 0 sets the field number to the first field.

Data Type

Data Type Long

C API TipcMsgSetCurrent, TipcMsgGetCurrent

Example This example sets the current field of the message:

```
Dim fieldNum As Long
'Determine field location in the message
fieldNum = msg.Current

'Point to the 3rd field of the message
msg.Current = 2

'Point to the very first field in the message
msg.Current = 0
```

CurrentUnknown Property

The CurrentUnknown property returns True if the value of the current field of a message is unknown or unset. This is a read-only property.

Syntax *object*.CurrentUnknown
where *object* is an object expression that evaluates to a TipcMsg object.

Data Type	Boolean
C API	TipcMsgGetCurrentFieldUnknown
Example	This example determines if the current field of the message is of unknown value: <pre>Dim fieldUnk As Boolean fieldUnk = msg.CurrentUnknown</pre>

DeliveryMode Property

The DeliveryMode property specifies the level of guarantee, if any, provided when a message is sent. By default, messages are sent without guaranteed message delivery (GMD).

Syntax	<code>object.DeliveryMode [=delivery_mode]</code> where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>delivery_mode</i> is an integer that specifies the delivery mode. Valid values are: • 0 is for <code>tipcDeliveryBestEffort</code>, the default setting. When a message is sent, no special action is taken to ensure delivery of the message. • 1 is for <code>tipcDeliverySome</code>, which is a GMD mode. The delivery of a message is successful when at least one of the specified receivers acknowledges delivery of the message. • 2 is for <code>tipcDeliveryAll</code>, which is a GMD mode. The delivery of a message is successful when all specified receivers acknowledge delivery of the message. • 3 is for <code>tipcDeliveryOrdered</code>. No special action is taken to ensure delivery of a message, but messages are received in the order they were published, even if a network or protocol failure occurs.

Data Type	TipcDeliveryMode
Remarks	For more information on delivery modes and GMD, see the <i>TIBCO SmartSockets User's Guide</i> .
C API	TipcMsgSetDeliveryMode, TipcMsgGetDeliveryMode

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example sets the delivery mode of the message to `tipcDeliveryAll`:

```
Dim mode As Integer

'Determine delivery mode
mode = msg.DeliveryMode

'Change delivery mode to ensure that all get the message
msg.DeliveryMode = tipcDeliveryAll
```

DeliveryTimeout Property

The `DeliveryTimeout` property specifies the time period to wait for GMD (see `DeliveryMode` Property). The message `DeliveryTimeout` property is used to override a connection `DeliveryTimeout` property.

Syntax `object.DeliveryTimeout [=timeout]`
where:
object is an object expression that evaluates to a `TipcMsg` object.
timeout is a double real number that specifies the timeout period, in seconds, to wait for GMD delivery.

Data Type Double

Remarks See the GMD chapter in the *TIBCO SmartSockets User's Guide* for more information about delivery modes and delivery timeouts.

C API `TipcMsgSetDeliveryTimeout`, `TipcMsgGetDeliveryTimeout`

Example This example sets the delivery timeout of the message:

```
Dim timeout As Double

'Increase the current timeout by 10 seconds
timeout = msg.DeliveryTimeout
timeout = timeout + 10.0
msg.DeliveryTimeout = timeout
```

Dest Property

The Dest property specifies the destination of a message. The meaning of this property depends on how the message is used. Publish-subscribe messages between RTclients and RTservers use a subject name for the message destination and sender. Other applications might use this property for other purposes. The destination property default is set to an empty string ("").

Syntax `object.Dest [=destination]`
 where:
object is an object expression that evaluates to a TipcMsg object.
destination is a string that specifies the subject to which the message is published. It is user-defined in other applications.

Data Type String

C API TipcMsgSetDest, TipcMsgGetDest

Example This example sets the destination property of the message:

```
Dim d As String

'Determine current destination setting
d = msg.Dest

'Send message to /stocks/financial/... subject
msg.Dest = "/stocks/financial/..."
```

Fields Property

The Fields property is an instance of the TipcMsgFields class, containing the message's data fields. This is a read-only property, and need not be changed. To manipulate the fields within, see TipcMsgFields Class on page 170.

Syntax `object.Fields`
 where *object* is an object expression that evaluates to a TipcMsg object.

Data Type TipcMsgFields

C API No comparable C API function

Example This example prints out the type of each field in the message:

```
Dim field As Variant
For Each field in msg.Fields
    Select Case VarType(field)
        Case vbInteger
            Debug.Print "Int2 Field "
        Case vbInteger + vbArray
            Debug.Print "Int2Array Field "
        Case vbLong
            Debug.Print "Int4 Field "
        Case vbLong + vbArray
            Debug.Print "Int4Array Field "
        Case vbSingle
            Debug.Print "Real4 Field "
        Case vbSingle + vbArray
            Debug.Print "Real4Array Field "
        Case vbDouble
            Debug.Print "Real8 Field "
        Case vbDouble + vbArray
            Debug.Print "Real8Array Field "
        Case vbString
            Debug.Print "Str Field"
        Case vbString + vbArray
            Debug.Print "StrArray Field"
        Case Else
            Debug.Print "Unexpected Field type"
    End Select
Next field
```

HeaderStrEncode Property

The HeaderStrEncode property enables the compression of the message header into four-byte integers, for each message.

If the message header string encode is not set, the message type header string encode setting is used.

Syntax *object*.HeaderStrEncode [=header_str_encode]

where:

object is an object expression that evaluates to a TipcMsg object.

header_str_encode is a Boolean value:

- True enables the header string encode for each message. True indicates that this message's header is encoded.
- False disables the header string encode for each message, and the message type header string encode setting is used for this message. This is the default value for the header string encode property.

Data Type	Boolean
Remarks	For more information, see the <i>TIBCO SmartSockets User's Guide</i> .
C API	TipcMsgSetHeaderStrEncode, TipcMsgGetHeaderStrEncode
Example	This example sets the header string encoding property of the message: <pre>'Invert the HeaderStrEncode property of this message Dim encode As Boolean Dim opposite As Boolean encode = msg.HeaderStrEncode if encode Then opposite = False Else opposite = True End if msg.HeaderStrEncode = opposite</pre>

LbMode Property

In publish-subscribe operations, the (load balancing mode) property specifies to which RTclient a message is routed.

If the message load balancing mode is not set, the message type load balancing mode is used. The default value for the message type load balancing is `tipcLbNone`.

Syntax	<code>object.LbMode [=lb_mode]</code> where: <code>object</code> is an object expression that evaluates to a <code>TipcMsg</code> object. <code>lb_mode</code> is an integer that specifies the load balancing mode. Valid values are: • 0 is for <code>tipcLbNone</code>, the default setting. The message is routed with no load balancing. The message is routed to all RTclients subscribing to the destination subject. • 1 is for <code>tipcLbRoundRobin</code>. The message is routed to each RTclient subscribing to the destination subject, in turn. • 2 is for <code>tipcLbWeighted</code>. The message is routed to the RTclient with the fewest number of unacknowledged messages. • 3 is for <code>tipcLbSorted</code>. The message is routed to the RTclient in the lowest lexicographical order (sorted by unique subject).

Data Type	TipcLbMode

Remarks	See the <i>TIBCO SmartSockets User's Guide</i> for detailed information about load balancing.
C API	TipcMsgSetLbMode, TipcMsgGetLbMode
Example	This example changes a message's load balancing mode to round-robin: <pre>'Change Load Balancing Mode to Round Robin msg.LbMode = tipcLbRoundRobin</pre>

NumFields Property

The NumFields property sets the number of fields in message by truncating the message to the desired number of fields. If set to 0, the message is emptied.

Syntax *object*.NumFields [=num_fields]
where:
object is an object expression that evaluates to a TipcMsg object.
num_fields is a long integer that specifies how many fields to keep in the message.

Data Type Long

C API TipcMsgGetNumFields, TipcMsgSetNumFields

Example This example retrieves and sets the number of fields in the message:

```
'Determine number of fields in a message  
numFields = msg.NumFields 'OR  
numFields = msg.Fields.Count  
  
'Truncate message so that we can reuse it  
msg.NumFields = 0
```

Priority Property

The Priority property sets the level of importance of a message. The greater the number specified, the higher the level of importance assigned to a message. When a message is sent, it is placed in the connection message queue of the receiving process in the order of priority (level of importance).

The message type priority is used, if set. If the message type priority is unknown, then the message priority default is Default_Msg_Priority which is 0.

Syntax	<code>object.Priority [=priority]</code> where: <i>object</i> is an object expression that evaluates to a <code>TipcMsg</code> object. <i>priority</i> is an integer that specifies the message priority level.
Data Type	Integer
Remarks	For more information on delivery modes and GMD, see the <i>TIBCO SmartSockets User's Guide</i> .
C API	<code>TipcMsgSetPriority</code> , <code>TipcMsgGetPriority</code>
Example	This example increases a message's priority by one: <pre>Dim pri as Integer pri = msg.Priority pri = pri + 1 msg.Priority = pri</pre>

Sender Property

The Sender property identifies the originator of a message. When the message is sent through publish-subscribe, the originator is the unique subject name of the sending process. The sender property is read-only.

Syntax	<code>object.Sender</code> where <i>object</i> is an object expression that evaluates to a <code>TipcMsg</code> object.
Data Type	String
Remarks	The Sender property cannot be set.
C API	<code>TipcMsgGetSender</code>
Example	This example determines the sender of a message: <pre>Dim s As String s = msg.Sender</pre>

SeqNum Property

The SeqNum property uniquely identifies the message for GMD so that duplicate messages can be detected by the receiver(s). When the message is sent with either the tipcDeliverySome or tipcDeliveryAll delivery modes, the sequence number of the message is set to a unique incremented number. Because the receiver records the sequence number when the message is received, if the message is later resent because of GMD failure, the receiver detects (and accepts) the resent message. The sequence number property is set automatically and cannot be set manually.

Syntax *object*.SeqNum
where *object* is an object expression that evaluates to a TipcMsg object.

Data Type Long

C API TipcMsgSetSeqNum, TipcMsgGetSeqNum

Example This example determines the sequence number of the message:

```
Dim num As Long
Dim msg As TipcMsg

Set msg = TipcSvc1.Srv.MsgNext(2.0)
if msg.isNull() = False Then
    num = msg.SeqNum
    msg.Destroy
End If
```

Type Property

The Type property identifies the kind of message being processed. This property must be supplied when creating a message.

Syntax *object*.Type [=*mt*]
where:
object is an object expression that evaluates to a TipcMsg object.
mt is class TipcMt, and specifies the predefined message type to which this message's format conforms.

Data Type TipcMt

Remarks See the information on message types in the *TIBCO SmartSockets User's Guide* for more information on the available message types.

C API TipcMsgSetType, TipcMsgGetType

Example This example changes the message type to numeric:

```
Dim msgName as String
msg.Type = TipcSvc1.MtLookup("numeric_data")
msgName = msg.Type.Name
```

UserProp Property

The UserProp property is a user-defined property and can be used for any purpose, such as attaching a version number to messages. This property is not used internally by SmartSockets. The default value is 0.

Syntax *object*.UserProp [=*user_prop*]

where:

object is an object expression that evaluates to a TipcMsg object.

user_prop is a long integer. This is a user-defined value. The default value is 0.

Data Type Long

C API TipcMsgSetUserProp, TipcMsgGetUserProp

Example This example adds 10 to the user property of a message:

```
Dim prop As Long
prop = msg.UserProp
prop = prop + 10
msg.UserProp = prop
```

TipcMsgFields Class

TipcMsgFields — user class for message field operations

Description	The TipcMsgFields class provides an interface to the fields contained in SmartSockets messages. It allows you to add, retrieve and count the fields contained within messages. For more information on message fields, see the <i>TIBCO SmartSockets User's Guide</i> .
Methods	• Add Method • AddUnknown Method
Properties	• Count Property • Item Property
Events	None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcMsgFields Class Methods

Add Method

The Add method appends data to a message, without specifying the data type. It is the simplest way to append data to a message, and can be used in place of the Append* or AddNamed* methods of the TipcMsg class. All the same data types that can be added with the Append* methods can be added with the Add method. As always, arrays must be one-dimensional.

Syntax *object*.Add *va_data* [, *name*]

where:

object is an object expression that evaluates to a TipcMsgFields object.

va_data is a variant that specifies the data to add to the message. If the data is an array, the array must be one-dimensional.

name This is optional and used only if you want to add a named field. It is a string that specifies the field name to add to the message.

Return Type None

C API TipcMsgAppend*, TipcMsgAddNamed*

Example This example shows how to add each type of field to a message:

```
Dim nmsg As New TipcMsg
Dim nmt As New TipcMt
Dim b As Byte
Dim b_array(3) As Byte
Dim i2 As Integer
Dim i2_array(4) As Integer
Dim i4 As Long
Dim i4_array(2) As Long
Dim r4 As Single
Dim r4_array(2) As Single
Dim r8 As Double
Dim r8_array(2) As Double
Dim s As String
Dim s_array(2) As String

b_array(0) = Asc("a")
b_array(1) = Asc("B")
b = Asc("z")
i2 = 12
i2_array(0) = 98
i2_array(1) = 99
i4 = 12345678
i4_array(0) = 1000
i4_array(1) = 2000
```

```
r4 = 1234.123
r4_array(0) = 123.123
r4_array(1) = 567.789
r8 = 12345678.123456
r8_array(0) = 1000.34
r8_array(1) = 2000.345
s = "This is a test string"
s_array(0) = "Another test string"
s_array(1) = "And yet another"
nmt.Lookup "simple" 'This message type has grammar of int2
nmsg.Create nmt
nmsg.Fields.Add 23

msg.Fields.Add b_array 'AppendBinary
msg.Fields.Add b 'AppendChar
msg.Fields.Add i2 'AppendInt2
msg.Fields.Add i2_array 'AppendInt2Array
msg.Fields.Add i4 'AppendInt4
msg.Fields.Add i4_array 'AppendInt4Array
msg.Fields.Add r4 'AppendReal4
msg.Fields.Add r4_array 'AppendReal4Array
msg.Fields.Add r8 'AppendReal8
msg.Fields.Add r8_array 'AppendReal8Array
msg.Fields.Add s 'AppendStr
msg.Fields.Add s_array 'AppendStrArray
msg.Fields.Add nmsg 'AppendMsg
msg.Fields.Add nmsg "Named_Message" 'AddNamedMsg
```

AddUnknown Method

The AddUnknown method is similar to the Add method, except that the field type is explicitly given, and the value of that field is "no value". This is useful in cases where the data may be missing, for example, from a stream of telemetry data, but the fact that it is missing is important.

Syntax *object*.Add *field_type*
 where:
 object is an object expression that evaluates to a TipcMsgFields object.
 field_type is an integer that specifies the field type.

Return Type None

C API TipcMsgAppendUnknown

Example This example shows how to add a field with an unknown value to a message:
 msg.Fields.AddUnknown tipcFtChar 'char type

TipcMsgFields Class Properties

Count Property

The Count property returns the number of fields in the TipcMsgFields object. It is a read-only value.

Syntax *object*.Count
 where *object* is an object expression that evaluates to a TipcMsgFields object.

Data Type Long

C API TipcMsgGetNumFields

Example This example determines the number of fields in a message:

```
Dim nf As Long
nf = msg.Fields.Count
```

Item Property

The Item property returns the variant data in the specified field. The first field is specified by index 0.

Syntax *object*.Item (*index*)
 where:
object is an object expression that evaluates to a TipcMsgFields object.
index is a long integer that specifies which field to return. A value of 0 specifies that the first field is returned.

Data Type Long

Remarks The word "item" has special meaning in most ActiveX environments. The Item keyword can be omitted and *index* applied directly to the object. For fields with unknown values, those added with AddUnknown, the special variant type VT_EMPTY is returned.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

C API No comparable function

Example This example illustrates the Item property used directly (A) and indirectly (B) to display the contents of a message's fields:

```
Dim msg as New TipcMsg

msg.Create (TipcSvc1.MtLookup("numeric_data"))
msg.Fields.Add 123.456
msg.Fields.Add 10.123
Debug.Print msg.Fields.Item(0)
' (A) debug window will display "123.456"
Debug.Print msg.Fields(1)
' (B) debug window will now also display "10.123"
```

TipcMt Class

TipcMt — user class for constructing, looking up and destroying message types

Description	The TipcMt class acts as an interface for creating message types. 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. For more information on message types, see the <i>TIBCO SmartSockets User's Guide</i> .
Methods	• Create Method • Destroy Method • Lookup Method • LookupByNum Method
Properties	• DeliveryMode Property • DeliveryTimeout Property • Grammar Property • HeaderStrEncode Property • LbMode Property • Name Property • Num Property • Priority Property • UserProp Property
Events	None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcMt Class Methods

Create Method

The Create method defines a new SmartSockets message type.

Syntax *object*.Create *name*, *num*, *grammar*
where:
object is an object expression that evaluates to a TipcMt object.
name is a string representing the name of the message type. It is not case-sensitive.
num is a long integer that specifies the numerical ID being assigned to the new message type.
grammar is a string that specifies the description of the message's format used for reading and writing message files.

Return Type None

Remarks See the *TIBCO SmartSockets User's Guide* for more information about grammar and field types, as well as numeric message IDs.

C API TipcMtCreate

Example This example creates and then destroys a new message type test_mt:

```
Dim mt as New TipcMt
mt.Create "test_mt", 3000, "str int4 int4 msg"
mt.Destroy
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Destroy Method

The Destroy method destroys a message type. When a message type is destroyed, all memory for the message type is freed. This also allows you to create new message types with the same numeric assignment in the place of the destroyed message type.

Syntax `object.Destroy`

where *object* is an object expression that evaluates to a TipcMt object.

Return Type None

C API TipcMtDestroy

Example This example creates multiple message types with the same number, then destroys the message types:

```
Dim mt As New TipcMt
Dim msg as TipcMsg

mt.Create "test_mt", 3000, "str int4 int4"
set msg = TipcSvc1.MsgCreate(mt)
msg.Fields.Add "Test string"
msg.Fields.Add 99999
msg.Fields.Add 12345
msg.Dest "tmp"
TipcSvc1.Srv.MsgSend msg
TipcSvc1.Srv.Flush

'Destroy the message type so that it can be reused
mt.Destroy
mt.Create "another mt", 3000, "int2 int2_array str"
set msg = TipcSvc1.MsgCreate(mt)
```

Lookup Method

The Lookup method initializes a TipcMt object to be a SmartSockets message of the specified type. The predefined message type is specified by its descriptive name.

Syntax `object.Lookup name`

where:

object is an object expression that evaluates to a TipcMt object.

name is a string representing the message type name to look up. It is not case-sensitive.

Return Type None

C API TipcMtLookup

Example This example looks up the `control` and `Info` message types:

```
Dim mt As New TipcMt
mt.Lookup "control"
mt.Lookup "Info"
```

LookupByNum Method

The `LookupByNum` method initializes a `TipcMt` object to be a `SmartSockets` message of the specified type. The predefined message type is specified by its numeric assignment.

Syntax *object.LookupByNum num*

where:

object is an object expression that evaluates to a `TipcMt` object.

num is a long integer that specifies the numeric ID of the `SmartSockets` message type to look up.

Return Type None

C API TipcMtLookupByNum

Example This example looks up message types by their associated number:

```
Dim mt As New TipcMt
mt.LookupByNum -30 'Control Message type
mt.LookupByNum -40 'Info Message Type
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcMt Class Properties

DeliveryMode Property

The DeliveryMode property specifies the default level of guarantee, if any, when a message of the TipcMt object's type is sent. By default, messages are sent without GMD.

Syntax `object.DeliveryMode [=default_mode]`

where:

`object` is an object expression that evaluates to a TipcMt object.

`default_mode` is a TipcDeliveryMode class that specifies the default delivery mode for all messages of this message type. Valid values are:

- 0 is for `tipcDeliveryBestEffort`, the default setting. When a message is sent, no special action is taken to ensure delivery of the message.
- 1 is for `tipcDeliverySome`, which is a GMD mode. The delivery of a message is successful when at least one of the specified receivers acknowledges delivery of the message.
- 2 is for `tipcDeliveryAll`, which is a GMD mode. The delivery of a message is successful when all specified receivers acknowledge delivery of the message.
- 3 is for `tipcDeliveryOrdered`. No special action is taken to ensure delivery of a message, but messages are received in the order they were published, even if a network or protocol failure occurs.

Data Type `TipcDeliveryMode`

Remarks For more information on delivery modes, see the GMD chapter in the *TIBCO SmartSockets User's Guide*.

C API `TipcMsgSetDeliveryMode`, `TipcMsgGetDeliveryMode`

Example This example creates a new message type that is sent with GMD:

```
Dim mt As TipcMt
Dim mode As Integer

mt.Create "test_mt", 3000, "int4 int4"
```

```
mt.DeliveryMode = tipcDeliveryAll
mode = mt.DeliveryMode
If mode <> tipcDeliveryAll Then
    Debug.Print "Unable to set Delivery Mode successfully"
End If
```

DeliveryTimeout Property

The DeliveryTimeout property identifies the time period to wait for GMD (see DeliveryMode Property). The message DeliveryTimeout property is used to override a connection DeliveryTimeout property.

Syntax *object*.DeliveryTimeout [=*timeout*]
where:
object is an object expression that evaluates to a TipcMt object.
timeout is a double real number that specifies the time, in seconds, to wait for GMD messages to be delivered.

Data Type Double

Remarks For more information, see the *TIBCO SmartSockets User's Guide*.

C API TipcMtSetDeliveryTimeout, TipcMtGetDeliveryTimeout

Example This example increases the delivery timeout for a message type by 10 seconds:

```
Dim mt As New TipcMt
Dim timeout As Double

mt.Create "new message", 1000, "int4 real8"
timeout = mt.DeliveryTimeout
timeout = timeout + 10
mt.DeliveryTimeout = timeout
```

Grammar Property

The Grammar property identifies the format for writing out the contents of a message to message files. The Grammar property is read-only.

Syntax *object*.Grammar
where *object* is an object expression that evaluates to a TipcMt object.

Data Type	String
Remarks	See the <i>TIBCO SmartSockets User's Guide</i> for information about grammar and message types.
C API	TipcMtGetGrammar
Example	This example retrieves the grammar of a monitoring message: <pre>Dim mt as New TipcMt Dim s As String Dim n As String mt.Lookup "mon_server_general_buffer_poll_result" s = mt.Grammar n = mt.Name Debug.Print "Grammar of " & n & " is <" & s & ">"</pre>

HeaderStrEncode Property

The HeaderStrEncode property specifies whether or not the compression of the message header into four-byte integers happens by default for this message type.

Syntax	<code>object.HeaderStrEncode [=header_str_encode]</code> where: <code>object</code> is an object expression that evaluates to a TipcMt object. <code>header_str_encode</code> is a Boolean value that specifies the default message header compression mode for messages of this type: • True means that by default, message headers are compressed. • False means that message headers are not compressed. This is the default value for the header string encode property.
Data Type	Boolean
Remarks	Setting the header string encode for each message can be useful, but setting the message type header string encode property changes the default behavior of all outgoing messages of that specific type. For more information, see the <i>TIBCO SmartSockets User's Guide</i> .
C API	TipcMtSetHeaderStrEncode, TipcMtGetHeaderStrEncode

Example This example displays text in the Debug window to indicate if header encoding is turned on by default for the message type used by the msg message:

```
mst = msg.Type.Name
If (msg.Type.HeaderStrEncode = True) Then
    Debug.Print "Encoding is On for type ",mst;
Else
    Debug.Print "Encoding is Off for type ",mst;
End If
```

LbMode Property

The LbMode property identifies to which RTclient a message is routed, depending on the load balancing mode specified. This value is used as the default load balancing mode for messages of this type.

Syntax

`object.LbMode [=default_lb_mode]`
where:
`object` is an object expression that evaluates to a TipcMt object.
`default_lb_mode` is class TipcLbMode and specifies the default load balancing mode for messages of this message type. Valid values are:

- `tipcLbNone`
means the message is routed with no load balancing. The message is routed to all RTclients subscribing to the destination subject.
- `tipcLbRoundRobin`
means the message is routed to each RTclient subscribing to the destination subject, in turn.
- `tipcLbSorted`
means the message is routed to the RTclient in the lowest lexicographical order (sorted by unique subject).
- `tipcLbWeighted`
means the message is routed to the RTclient with the fewest number of unacknowledged messages.

Data Type TipcLbMode

Remarks See the *TIBCO SmartSockets User's Guide* for detailed information about load balancing.

C API TipcMtSetLbMode, TipcMtGetLbMode

Example This example prints out the load balancing mode:

```
mst = msg.Type.Name
Select Case msg.Type.LbMode
Case tipcLbNone
    Debug.Print "No load balancing for " & mst
Case tipcLbRoundRobin
    Debug.Print "Round Robin load balancing for " & mst
Case tipcLbSorted
    Debug.Print "Sorted load balancing for " & mst
Case tipcLbWeighted
    Debug.Print "Weighted load balancing for " & mst
End Select
```

Name Property

The Name property returns the name of the message type specified by the object. It is a read-only value and is set when the message type is created.

Syntax *object*.Name

where *object* is an object expression that evaluates to a TipcMt object.

Data Type String

C API TipcMtGetName

Example This example displays the name of the type of message:

```
Debug.Print "The variable 'msg' contains a message  
              of type "; msg.Type.Name
```

Num Property

The Num property returns the assigned numeric ID of the message type specified by the object. It is a read-only value and is set when the message type is created.

Syntax *object*.Num

where *object* is an object expression that evaluates to a TipcMt object.

Data Type Long

C API TipcMtGetNum

Example This example displays the identifying number of the message type:

```
Debug.Print "The variable 'msg' contains a message  
with type ID="; msg.Type.Num
```

Priority Property

The Priority property identifies the default level of importance for messages of this message type.

Syntax *object*.Priority [=priority]
where:
object is an object expression that evaluates to a TipcMt object.
priority is an integer that specifies the default priority for messages of this message type.

Data Type Integer

Remarks The greater the number specified, the higher the level of importance assigned to a message. When a message is sent, it is placed in the connection message queue of the receiving process in the order of priority. If the message type priority is not set, then the message priority default is Default_Msg_Priority, which is 0.

C API TipcMtSetPriority, TipcMtGetPriority

Example This example sets the priority property of the message type:

```
Dim mt as New TipcMt  
mt.Create("priority_test",2002,"int4")  
mt.Priority = 5  
Debug.Print "Newly created message type's priority is";  
mt.Priority  
mt.Destroy
```

UserProp Property

The UserProp is a user-defined property and can be used for any purpose, such as attaching a version number to messages. This property is not used internally by SmartSockets. The user property is 0 by default.

Syntax *object*.UserProp [=user_prop_value]

where:

object is an object expression that evaluates to a TipcMt object.

user_prop_value is a long integer that specifies the default user-defined property for messages of this message type.

Data Type Long

C API TipcMtSetUserProp, TipcMtGetUserProp

Example This example sets the user property of a new message type:

```
Dim mt As New TipcMt
Dim prop as Long

mt.Create "new message", 1000, "int4"
mt.UserProp = 100
prop = mt.UserProp
If prop <> 100 Then
    Debug.Print "Unable to set User Property"
End If
```

TipcProcessCb Class

TipcProcessCb — class for handling SmartSockets messages arriving to your program from the TipcMsg event

Description	The TipcProcessCb component contains the TipcProcessCb class, providing the TipcProcessCb event that invokes when a SmartSockets message is delivered to your program. The TipcProcessCb object must first be activated by adding it to a TipcConn or TipcSrv object. See the AddProcessCb and AddDefaultCb methods of those classes for more information. For more information on callbacks, see the <i>TIBCO SmartSockets User's Guide</i> .
Methods	None
Properties	None
Events	TipcMsg

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcProcessCb Class Events

TipcMsg Event

The TipcMsg event is invoked when a SmartSockets message is delivered to an RTclient. Place message-handling code in this event.

Syntax `Private Sub object_TipcMsg (conn, msg,param)`
 where:
 object is an object expression that evaluates to a TipcProcessCb object.
 conn is class TipcConn and specifies the connection on which the message arrives.
 msg is class TipcMsg and is the actual SmartSockets message that is received.
 param is optional and is an *ole_variant* that specifies a user-defined argument set when registering the callback. See TipcSrv.AddProcessCb, or TipcSrv.AddSubjectCb, or TipcSrv.AddDefaultCb.

Return Type None

Remarks You must register the TipcProcessCb event with the AddDefaultCb, AddProcessCb, AddSubjectCb or AddSubjectDefaultCb methods for the TipcMsg event to be invoked.

C API Comparable to user-defined functions installed with TipcSrvProcessCbCreate, TipcSrvDefaultCbCreate, TipcSrvSubjectCbCreate and TipcSrvSubjectDefaultCbCreate.

Example This example displays every field of every message it receives:

```
Dim tmsg as TipcMsg
Dim nmsg As TipcMsg
Dim field As Variant
Dim i As Long

Set tmsg = msg
For Each field in tmsg
    if VarType(field) > vbArray Then
        Debug.Print TypeName(field) & " received"
        For i = LBound(field) to UBound(field) - 1
            Debug.Print Format(field(i))
        Next i
    Else
        If VarType(field) = vbObject Then
            set nmsg = field
            Debug.Print nmsg.Type.Name & " message received"
```

```
        Else
            Debug.Print TypeName(field) & ": " & Format(field)
        End If
    End If
Next field
```


TipcSrv Class

TipcSrv — class for sending and receiving messages with RTserver and registering interests in subjects

Description The TipcSrv class provides an interface to the function to send, receive and process SmartSockets messages, as well as manage RTclient's publish-subscribe participation. For more information on RTserver, see the *TIBCO SmartSockets User's Guide*.

Methods

- AddDefaultCb Method
- AddProcessCb Method
- AddSubjectDefaultCb Method
- Create Method
- Flush Method
- GmdFileDelete Method
- GmdMsgResend Method
- GmdMsgStatus Method
- KeepAlive Method
- LogRemoveMt Method
- MsgInsert Method
- MsgProcess Method
- MsgSend Method
- MsgSendRpc Method
- RemoveErrorCb Method
- RemoveSubjectCb Method
- RemoveSubjectDefaultCb Method
- StdSubjectSubscribe Method
- SubjectLbInit Method
- AddErrorCb Method
- AddSubjectCb Method
- Check Method
- Destroy Method
- GmdFileCreate Method
- GmdMsgDelete Method
- GmdMsgServerDelete Method
- GmdResend Method
- LogAddMt Method
- MainLoop Method
- MsgNext Method
- MsgSearchType Method
- MsgSendEx Method
- RemoveDefaultCb Method
- RemoveProcessCb Method
- RemoveSubjectCbAll Method
- StdSubjectRetrieve Method
- SubjectGmdInit Method
- SubjectRetrieve Method

Properties

- Arch Property
- BlockMode Property
- GmdDir Property
- GmdNumPending Property
- AutoFlushSize Property
- ConnStatus Property
- GmdMaxSize Property
- Lcn Property

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

	• Node Property • Pid Property • Timeout Property • User Property	• NumQueued Property • Subjects Property • UniqueSubject Property
Events	None	

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcSrv Class Methods

Accept Method

The Accept method is not implemented in the TipcSrv object. It is present because TipcSrv's methods are inherited from TipcConn, which does have this function.

AddDefaultCb Method

The AddDefaultCb method installs an event handler for SmartSockets messages that arrive at RTclient. The default callback is used when there is no process callback for the particular type of message being received.

Syntax	<code>object.AddDefaultCb event_handler [,priority, param]</code> where: <code>object</code> is an object expression that evaluates to a TipcSrv object. <code>event_handler</code> evaluates to a TipcProcessCb object whose TipcMsg event is invoked when a message with no other applicable handler is received. <code>priority</code> is an integer that specifies the priority of this handler. The higher priority handlers of the same type are called first. If not specified, the default priority is 1. This is an optional parameter. <code>param</code> is optional and is an <code>ole_variant</code> that specifies a user-defined argument accessible when a callback function is invoked. This parameter is useful to avoid redundant code. For example, if updating several cells in a spread sheet using the same callback function, this parameter could be used as the cell to update.
Return Type	None
Remarks	See the <i>TIBCO SmartSockets User's Guide</i> for more information on default callbacks.
C API	TipcSrvDefaultCbCreate

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example installs a default event handler:

```
Dim s as TipcSrv
Set s = TipcSvc1.Srv
s.AddDefaultCb TipcProcessCb1.Object
```

AddErrorCb Method

The AddErrorCb method installs an event handler for unrecoverable SmartSockets errors.

Syntax `object.AddErrorCb event_handler [,priority, param]`
where:
object is an object expression that evaluates to a TipcSrv object.
event_handler evaluates to a TipcErrorCb object whose TipcMsg event is invoked when an unrecoverable SmartSockets error is encountered.
priority is an integer that specifies the priority of this handler. The higher priority handlers of the same type are called first. If not specified, the default priority is 1. This is an optional parameter.
param is optional and is an *ole_variant* that specifies a user-defined argument accessible when a callback function is invoked. This parameter is useful to avoid redundant code. For example, if updating several cells in a spread sheet using the same callback function, this parameter could be used as the cell to update.

Return Type None
Remarks See the *TIBCO SmartSockets User's Guide* for more information on error callbacks.
C API TipcSrvErrorCbCreate

Example This example installs an error handler. Drag a TipcErrorCb icon onto your form, name the object TipcErrCb1 and add this code:

```
Dim s as TipcSrv
Set s = TipcSvc1.Srv
s.AddErrorCb TipcErrorCb1.Object
```

AddProcessCb Method

The AddProcessCb method installs an event handler to process SmartSockets messages of a specific type. The TipcMsg event of the TipcProcessCb object specified is invoked when the RTclient receives a message of the given type.

Syntax `object.AddProcessCb event_handler, message_type [,priority, param]`

where:

object is an object expression that evaluates to a TipcSrv object.

event_handler evaluates to a TipcProcessCb object whose TipcMsg event is invoked when a message of *message_type* is received.

message_type evaluates to a TipcMt object that specifies the type of message that this process event handles.

priority is an integer that specifies the priority of this handler. The higher priority handlers of the same type are called first. If not specified, the default priority is 1. This is an optional parameter.

param is optional and is an `ole_variant` that specifies a user-defined argument accessible when a callback function is invoked. This parameter is useful to avoid redundant code. For example, if updating several cells in a spread sheet using the same callback function, this parameter could be used as the cell to update.

Return Type None

Remarks Do not mix process and subject callbacks in the same RTclient. They have different priority sets and mixing them gives indeterminate results, as priorities between callback types are not honored. For more information on Process callbacks, see the *TIBCO SmartSockets User's Guide*.

C API TipcSrvProcessCbCreate

Example This example adds a process callback for info messages. Drag a TipcProcessCb icon on to your form, name the object TipcProcessCb1 and add the code:

```
Dim s as TipcSrv, mtInfo as new TipcMt
Set s = TipcSvc1.Srv
mtInfo.Lookup("info")
s.AddProcessCb TipcProcessCb1.Object, mtInfo
```

AddSubjectCb Method

The AddSubjectCb method installs an event handler to process SmartSockets messages sent to a specified subject with a specified message type. The TipcMsg event of the TipcProcessCb object specified is invoked when a message of the given subject and type is received.

Syntax `object.AddSubjectCb event_handler, subject, message_type [,priority, param]`
where:
object is an object expression that evaluates to a TipcSrv object.
event_handler evaluates to a TipcProcessCb object whose TipcMsg event is invoked when a message of *message_type* is received by *subject*.
subject is a string that specifies the subject for which to create a callback. If *subject* is null, callbacks are created for all subjects.
message_type evaluates to a TipcMt object that specifies the type of message that this process event handles. If *message_type* is null, the process event handles all message types.
priority is an integer that specifies the priority of this handler. The higher priority handlers of the same type are called first. If not specified, the default priority is 1. This is an optional parameter.
param is optional and is an ole_variant that specifies a user-defined argument accessible when a callback function is invoked. This parameter is useful to avoid redundant code. For example, if updating several cells in a spread sheet using the same callback function, this parameter could be used as the cell to update.

Return Type None

Remarks Subject callbacks support the use of wildcards (* or ...). For more information on wildcards in subjects and subject callbacks, see the *TIBCO SmartSockets User's Guide*.

C API TipcSrvSubjectCbCreate

Example This example creates callbacks for all info messages, all messages, and messages with the subject "/stocks":

Dim s as TipcSrv, mtInfo as new TipcMt

Set s = TipcSvc1.Srv
mtInfo.Lookup("info")

'Create subject callback for all info messages received
s.AddSubjectCb TipcProcessCb1.Object, "/...", mtInfo

'Create subject callback for ALL messages received

```
s.AddSubjectCb TipcProcessCb2.Object, Null, Null
'Create subject cb for all info messages recvd by "/stocks"
s.AddSubjectCb TipcProcessCb3.Object, "/stocks", mtInfo
```

AddSubjectDefaultCb Method

The AddSubjectDefaultCb method installs an event handler to process SmartSockets messages that have no corresponding subject callback. The TipcMsg event of the TipcProcessCb object specified is invoked when a message not processed by any other subject callback is received.

Syntax	<code>object.AddSubjectDefaultCb event_handler [,priority, param]</code> where: <code>object</code> is an object expression that evaluates to a TipcSrv object. <code>event_handler</code> evaluates to a TipcProcessCb object whose TipcMsg event is invoked when a message is not processed by any other subject callbacks. <code>priority</code> is an integer that specifies the priority of this handler. The higher priority handlers of the same type are called first. If not specified, the default priority is 1. This is an optional parameter. <code>param</code> is optional and is an <code>ole_variant</code> that specifies a user-defined argument accessible when a callback function is invoked. This parameter is useful to avoid redundant code. For example, if updating several cells in a spread sheet using the same callback function, this parameter could be used as the cell to update.
Return Type	None
Remarks	For more information on default subject callbacks, see the <i>TIBCO SmartSockets User's Guide</i> .
C API	TipcSrvSubjectDefaultCbCreate
Example	This example adds a default subject callback: <pre>Dim s as TipcSrv Set s = TipcSvc1.Srv s.AddSubjectDefaultCb TipcProcessCb1.Object, 2</pre>

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Check Method

The Check method checks to see if data can be read from or written to the connection to an RTserver. This is a low-level function that typically is not used directly by developers.

Syntax *object*.Check (*check_mode*, *timeout*)

where:

object is an object expression that evaluates to a TipcSrv object.

check_mode is class TipcCheckMode, and specifies the mode to check for availability:

- 0 is tipcCheckRead and means check for data read availability
- 1 is tipcCheckWrite and means check for data write availability

timeout is a double real number that specifies how long, in seconds, to wait before returning a False value.

Return Type Boolean, where:

True indicates data can be read from or written to the connection to an RTserver.

False indicates data cannot be read from or written to an RTserver within *timeout* seconds.

C API TipcSrvCheck

Example This example checks to see if data is available for reading from the RTserver in three seconds:

```
If (Not TipcSvc1.Srv.Check(TipcCheckRead, 3.0)) Then
    Debug.Print "Can't read!"
End If
```

Create Method

The Create method creates a connection to an RTserver.

Syntax *object*.Create

where *object* is an object expression that evaluates to a TipcSrv object.

Return Type None

Remarks	To create a warm connection, set the TipcSrv.ConnStatus property to tipcSrvConnWarm, for example, by using: <pre>Dim TipcSrv as srv srv.ConnStatus := tipcSrvConnWarm</pre>
C API	TipcSrvCreate
Example	This example makes a connection to an RTserver: <pre>Dim s as TipcSrv Set s = TipcSvc1.Srv s.Create</pre>

Destroy Method

	The Destroy method terminates the connection to an RTserver.
Syntax	<i>object</i>.Destroy where <i>object</i> is an object expression that evaluates to a TipcSrv object.
Return Type	None
C API	TipcSrvDestroy
Example	This example terminates an established connection to an RTserver: <pre>Dim s as TipcSrv Set s = TipcSvc1.Srv s.Destroy</pre>

Flush Method

	The Flush method flushes the buffered outgoing messages on the connection to an RTserver.
Syntax	<i>object</i>.Flush where <i>object</i> is an object expression that evaluates to a TipcSrv object.
Return Type	None

Remarks	Flushing buffered messages can significantly influence performance of an application. While there are occasions to flush immediately after a message is sent, if this is done after every message, network bandwidth is wasted and throughput is diminished dramatically. Rather than explicitly flushing messages on a regular basis, use the AutoFlushSize property to tune the frequency of flushing for your application's message sizes and frequency.
C API	TipcSrvFlush
Example	This example flushes the connection to an RTserver: <pre>Dim s as TipcSrv Set s = TipcSvc1.Srv s.Flush</pre>

GmdFileCreate Method

The GmdFileCreate method creates a GMD area for the connection to an RTserver.

Syntax	<i>object</i> .GmdFileCreate where <i>object</i> is an object expression that evaluates to a TipcSrv object.
Return Type	None
Remarks	This is normally handled automatically, but is included to facilitate fine-grained GMD error control. For more information on GMD areas, see the <i>TIBCO SmartSockets User's Guide</i> .
C API	TipcSrvGmdFileCreate
Example	This example creates the GMD file area before sending the first GMD message: <pre>Dim msg As New TipcMsg TipcSvc1.Srv.GmdFileCreate msg.Create TipcSvc1.MtLookup("info") msg.DeliveryMode = tipcDeliveryAll TipcSvc1.Srv.MsgSend TipcSvc1.Srv.Flush</pre>

GmdFileDelete Method

The GmdFileDelete method deletes a GMD area that was created with the GmdFileCreate method.

Syntax `object.GmdFileDelete`

where *object* is an object expression that evaluates to a TipcSrv object.

Return Type None

Remarks For more information on the GMD area, see the *TIBCO SmartSockets User's Guide*.

C API TipcSrvGmdFileDelete

Example This example removes a GMD area before sending a message:

```
Dim msg As New TipcMsg
TipcSvc1.Srv.GmdFileDelete
msg.Create TipcSvc1.MtLookup("info")
msg.DeliveryMode = tipcDeliveryBestEffort
TipcSvc1.Srv.MsgSend
TipcSvc1.Srv.Flush
```

GmdMsgDelete Method

The GmdMsgDelete method deletes a guaranteed message from the GMD area after a GMD failure.

Syntax `object.GmdMsgDelete msg`

where:

object is an object expression that evaluates to a TipcSrv object.

msg is class TipcMsg and is the message to delete from the GMD area.

Return Type None

Remarks For more information on deleting a message from the GMD area, see the *TIBCO SmartSockets User's Guide*.

C API TipcSrvGmdMsgDelete

Example This example deletes a GMD message:

```
TipcSvc1.Srv.GmdMsgDelete msg
```

GmdMsgResend Method

The GmdMsgResend method resends a GMD message after a GMD failure.

Syntax `object.GmdMsgResend msg`
where:
`object` is an object expression that evaluates to a TipcSrv object.
`msg` is class TipcMsg and is the message to be resent.

Return Type None

Remarks For more information on resending a failed GMD message, see the *TIBCO SmartSockets User's Guide*.

C API TipcSrvGmdMsgResend

Example This example resends a GMD message:
`TipcSvc1.Srv.GmdMsgResend msg`

GmdMsgServerDelete Method

The GmdMsgServerDelete method deletes a GMD message on the RTserver after a GMD failure.

Syntax `object.GmdMsgServerDelete msg`
where:
`object` is an object expression that evaluates to a TipcSrv object.
`msg` is class TipcMsg and is the message to delete from the RTserver's GMD area.

Return Type None

Remarks For more information on deleting a failed GMD message from RTserver, see the *TIBCO SmartSockets User's Guide*.

C API TipcSrvGmdMsgServerDelete

Example This example deletes a GMD message on RTserver:
`TipcSvc1.Srv.GmdMsgServerDelete msg`

GmdMsgStatus Method

The GmdMsgStatus method polls RTserver for the GMD status of a message. After this call, the RTserver sends a message of type T_MT_GMD_STATUS_RESULT back to the calling RTclient, which can be processed for the names of RTclients that have acknowledged receipt of the specified message.

Syntax `object.GmdMsgStatus (msg)`
where:
object is an object expression that evaluates to a TipcSrv object.
msg is class TipcMsg and is the message to delete from the GMD area.

Return Type None

C API TipcSrvGmdMsgStatus

Example This example creates a callback to handle the response message, `gmd_status_result`, from this method:

```
Dim mt As New TipcMt
mt.Lookup "gmd_status_result"
TipcSvc1.Srv.AddProcessCb mt
TipcSvc1.Srv.GmdMsgStatus msg
```

GmdResend Method

The GmdResend method resends all GMD messages after a delivery failure.

Syntax `object.GmdResend`
where *object* is an object expression that evaluates to a TipcSrv object.

Return Type None

Remarks For more information, see the *TIBCO SmartSockets User's Guide*.

C API TipcSrvGmdResend

Example This example resends all failed GMD messages:
`TipcSvc1.Srv.GmdResend`

KeepAlive Method

The KeepAlive method checks to determine if the connection to an RTserver is still available.

Syntax *object*.KeepAlive
 where *object* is an object expression that evaluates to a TipcSrv object.

Return Type Boolean value, where:
 True indicates the connection to the RTserver is intact.
 False indicates the connection to the RTserver was lost.

Remarks Upon returning False, the global SmartSockets error number is set and any applicable TipcError callback events are invoked.

C API TipcSrvKeepAlive

Example This example makes sure that the connection to the RTserver is still intact:

```
Dim alive As Boolean

alive = TipcSvc1.Srv.KeepAlive
If alive = False Then
    MsgBox "Connection is broken!", 0, "No RTserver"
End If
```

LogAddMt Method

The LogAddMt method adds a message type to a message file-logging type. User defined messages must call this method in order for the message type to be logged with the logging options. Those logging options are Log_In_Data, Log_In_Internal, Log_In_Status, Log_Out_Data, Log_Out_Internal and Log_Out_Status.

Syntax *object*.LogAddMt (*log_type*, *mt*)
 where:
 object is an object expression that evaluates to a TipcSrv object.

log_type is class TipcSrvLogType, and specifies the message file-logging type:

- 0 is `tipcSrvLogData` and specifies variable value message types such as `TIME` and `NUMERIC_DATA`
- 1 is `tipcSrvLogStatus` and specifies message types such as `ALERT` and `POPUP`
- 2 is `tipcSrvLogInternal` and specifies all other message types, such as `SUBJECT_SET_SUBSCRIBE` and `SERVER_STOP_CALL`

mt is class TipcMt which specifies the message type to add for logging.

Return Type	None
Remarks	The LogAddMt method can only be called before the connection to the RTserver has been established. For more information on logging, see the <i>TIBCO SmartSockets User's Guide</i> .
C API	TipcSrvLogAddMt
Example	This example creates new message types that can be logged:<pre>Dim mt1 As New TipcMt Dim mt2 As New TipcMt. mt1.Create "first", 1000, "int4 str" mt2.Create "second", 1010, "real8 int4" TipcSvc1.Srv.LogAddMt tipcSrvLogData mt1 TipcSvc1.Srv.LogAddMt tipcSrvLogData mt2 TipcSvc1.ParseStr "setopt log_in_data in.dat" TipcSvc1.ParseStr "setopt log_out_data out.dat"</pre>

LogRemoveMt Method

The LogRemoveMt method removes a message type from a message file-logging type that has previously been added with LogAddMt.

Syntax	<i>object</i>.LogRemoveMt <i>log_type</i>, <i>mt</i> where: <i>object</i> is an object expression that evaluates to a TipcSrv object.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

log_type is class `TipcSrvLogType`, and specifies the message file-logging type:

- 0 is `tipcSrvLogData` and specifies variable value message types such as `TIME` and `NUMERIC_DATA`
- 1 is `tipcSrvLogStatus` and specifies message types such as `ALERT` and `POPUP`
- 2 is `tipcSrvLogInternal` and specifies all other message types, such as `SUBJECT_SET_SUBSCRIBE` and `SERVER_STOP_CALL`

mt is class `TipcMt` which specifies the message type to remove for logging.

Return Type None

C API `TipcSrvLogRemoveMt`

Example This example removes message types from logging:

```
Dim mt1 As New TipcMt
Dim mt2 As New TipcMt

mt1.Lookup "first"
mt2.Lookup "second"
TipcSvc1.Srv.LogRemoveMt tipcSrvLogData mt1
TipcSvc1.Srv.LogRemoveMt tipcSrvLogData mt2
```

MainLoop Method

The `MainLoop` method reads and processes the `SmartSockets` messages present on the connection to an `RTserver`. This is a convenience function that can almost always be used in place of the combination of `MsgNext`, `MsgProcess` and `MsgDestroy`.

Syntax `object.MainLoop (timeout)`
where:
object is an object expression that evaluates to a `TipcSrv` object.
timeout is a double real number that specifies the maximum time in seconds to wait for messages to arrive. Specify 0 to perform one round of processing and return immediately.

Return Type None

Remarks The `AutoMainLoop` property can be set, which automatically invokes the `MainLoop` method's function. If you use `AutoMainLoop`, you need not use `MainLoop`.

C API TipcSrvMainLoop

Example This example processes all queued messages:
TipcSvc1.Srv.MainLoop 0.0

MsgInsert Method

The MsgInsert method inserts a message into the queue of messages on the connection to an RTserver, at the specified position.

Syntax *object*.MsgInsert *msg*, *position*
where:
object is an object expression that evaluates to a TipcSrv object.
msg is class TipcMsg and is the message to be inserted into the queue.
position is a long integer that specifies where (which position) in the queue to insert the message. Specify 0 to insert the message into the first position in the queue.

Return Type None

Remarks The message must not be destroyed by calling the MsgDestroy method, as MsgInsert takes responsibility for destroying the message.

C API TipcSrvMsgInsert

Example This example inserts a message at the front of the message queue:
Dim msg As New TipcMsg
msg.Create TipcSvc1.MtLookup("info")
msg.AppendStr "Informational string"
TipcSvc1.Srv.MsgInsert msg, 0

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

MsgNext Method

The MsgNext method retrieves the next message sent to an RTclient by RTserver.

Syntax *object*.MsgNext (*timeout*)
where:
object is an object expression that evaluates to a TipcSrv object.
timeout is a double real number that specifies the maximum time, in seconds, to wait for messages to arrive.

Return Type TipcMsg

Remarks The message returned by MsgNext should be destroyed when it is no longer needed.

C API TipcSrvMsgNext

Example This example retrieves a message in the queue, or one arriving in the next two seconds:

```
Dim msg as TipcMsg
Set msg = TipcSvc1.Srv.MsgNext(2.0)
TipcSvc1.Srv.MsgProcess msg 'Process message
msg.Destroy 'Destroy message
if msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
msg.Destroy
End If
```

MsgProcess Method

The MsgProcess method processes the specified message by firing any registered callback events, including default callbacks.

Syntax *object*.MsgProcess *msg*
where:
object is an object expression that evaluates to a TipcSrv object.
msg is class TipcMsg and is the message to process.

Return Type None

C API TipcSrvMsgProcess

Example This example retrieves a message and processes it:

```
Dim msg as TipcMsg
Set msg = TipcSvc1.Srv.MsgNext(2.0)
TipcSvc1.Srv.MsgProcess msg 'Process message
msg.Destroy 'Destroy message
```

MsgSearchType Method

The MsgSearchType method returns the first (by priority) message in the message queue of the specified type or waits up to the specified amount of time for a message of the desired type to arrive, and then returns that message.

Syntax *object*.MsgSearchType (*timeout*, *message_type*)

where:

object is an object expression that evaluates to a TipcSrv object.

timeout is a double real number that specifies the maximum time in seconds to wait for messages of *message_type* to arrive, if one is not already in the queue.

message_type is class TipcMt and specifies the message type to retrieve.

Return Type TipcMsg

Remarks The message returned by MsgSearchType should be destroyed when it is no longer needed.

C API TipcSrvMsgSearchType

Example This example processes the first info message received within the next 10 seconds:

```
Dim mt As New TipcMt
Dim msg as TipcMsg

mt.Lookup "info"
set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
if msg.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg
    msg.Destroy
End If
```

MsgSend Method

The MsgSend method sends the specified message to an RTserver. This is how RTclient publishes messages to subjects.

Syntax *object.MsgSend msg*
where:
object is an object expression that evaluates to a TipcSrv object.
msg is class TipcMsg and is the message to be published.

Return Type None

Remarks The value of the parent's TipcSvc's ServerMsgSend property controls whether this message is sent.

C API TipcSrvMsgSend

Example This example sends a quit force message to RTserver terminating all RTclients in the same project:

```
Dim msg as New TipcMsg
msg.Create TipcSvc1.MtLookup("control")
msg.AppendStr "quit force"
msg.Dest = "/..."
TipcSvc1.Srv.MsgSend
TipcSvc1.Srv.Flush
```

MsgSendEx Method

The MsgSendEx method sends the specified message to an RTserver. The exception is the special case where the parent TipcSvc's ServerMsgSend property is False, and the second parameter of this function is True, then the message is not sent, but only simulated.

Syntax *object.MsgSendEx msg, use_server_msg_send*
where:
object is an object expression that evaluates to a TipcSrv object.
msg is class TipcMsg and is the message to be published.

use_server_msg_send is a Boolean value:

- True means check the value of the ServerMsgSend property before transmitting. If that ServerMsgSend is False, simulate sending the message.
- False means ignore the value of the ServerMsgSend property and send the message.

Return Type	None
C API	TipcSrvMsgSend
Example	This example sends messages using the ServerMsgSend property: <pre>Dim s As TipcSrv Set s = TipcSvc1.Srv TipcSvc1.ServerMsgSend = False s.MsgSendEx msg, True 'Msg will not be sent s.MsgSendEx msg, False ' Msg will be sent TipcSvc1.ServerMsgSend = True s.MsgSendEx msg, True 'Msg will be sent s.MsgSendEx msg, False ' Msg will be sent</pre>

MsgSendRpc Method

The MsgSendRpc method performs a remote procedure call (RPC) with messages, from RTserver. The specified message is sent to the server and program operation halts until the timeout expires or the response message is received.

Syntax	<i>object</i>.MsgSendRpc (<i>msg</i>, <i>timeout</i>) where: <i>object</i> is an object expression that evaluates to a TipcSrv object. <i>msg</i> is class TipcMsg and is the RPC message to be sent. <i>timeout</i> is a double real number that specifies the maximum time in seconds to wait for the RPC response message to return.
Return Type	TipcMsg
Remarks	The numeric message ID of the RPC response must be equal to one greater than the ID of the RPC request.
C API	TipcSrvMsgSendRpc

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example sends a request message and waits to 30 seconds for a response to come back:

```
Dim mt1 As New TipcMt
Dim mt2 As New TipcMt
Dim msg1 As New TipcMsg
Dim msg2 As TipcMsg

mt1.Create "request", 1050, ""
mt2.Create "response", 1051, "int4"
msg1.Create mt1
msg1.Dest "/requestor"
Set msg2 = TipcSvc1.Srv.MsgSendRpc msg1, 30.0
If msg2.IsNull = False Then
    TipcSvc1.Srv.MsgProcess msg2
    msg2.Destroy
End If
msg1.Destroy
```

RemoveDefaultCb Method

The RemoveDefaultCb method uninstalls a default event handler installed with the AddDefaultCb method.

Syntax *object.RemoveDefaultCb event_handler*
where:
object is an object expression that evaluates to a TipcSrv object.
event_handler evaluates to a TipcProcessCb object whose default TipcMsg event is to be uninstalled.

Return Type None

C API TipcSrvDefaultCbLookup, TutCbDestroy

Example This example removes a default event handler:

```
Dim s as TipcSrv
Set s = TipcSvc1.Srv
s.RemoveDefaultCb TipcProcessCb1.Object
```

RemoveErrorCb Method

The RemoveErrorCb method uninstalls an event handler for unrecoverable errors that occur with the connection to an RTserver.

Syntax	<i>object.RemoveErrorCb event_handler</i> where: <i>object</i> is an object expression that evaluates to a TipcSrv object. <i>event_handler</i> evaluates to a TipcErrorCb object whose TipcMsg event is invoked when an unrecoverable RTserver connection error is encountered.
Return Type	None
C API	TipcSrvErrorCbLookup, TutCbDestroy
Example	This example removes an error event handler: <pre>Dim s as TipcSrv Set s = TipcSvc1.Srv s.RemoveErrorCb TipcErrorCb1.Object</pre>

RemoveProcessCb Method

The RemoveProcessCb method uninstalls an event handler that processes SmartSockets messages of a specific type. These event handlers are installed with the AddProcessCb method.

Syntax	<i>object.RemoveProcessCb event_handler, message_type</i> where: <i>object</i> is an object expression that evaluates to a TipcSrv object. <i>event_handler</i> evaluates to a TipcProcessCb object whose TipcMsg event is invoked when a message of <i>message_type</i> is received. <i>message_type</i> evaluates to a TipcMt object that specifies the type of message that this process event handles.
Return Type	None
C API	TipcSrvProcessCbLookup, TutCbDestroy

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example removes a process callback for info messages:

```
Dim s as TipcSrv, mtInfo as new TipcMt
Set s = TipcSvc1.Srv
mtInfo.Lookup("info")
s.RemoveProcessCb TipcProcessCb1.Object, mtInfo
```

RemoveSubjectCb Method

The RemoveSubjectCb method uninstalls an event handler that processes SmartSockets messages of a specific subject and specific type. These event handlers are installed with the AddSubjectCb method.

Syntax *object.RemoveSubjectCb event_handler, subject, message_type*
where:
object is an object expression that evaluates to a TipcSrv object.
event_handler evaluates to a TipcProcessCb object whose TipcMsg event is invoked when a message of *message_type* is received by *subject*.
subject is a string that specifies the subject that this event handler processes. If *subject* is null, the event handler processes all subjects.
message_type evaluates to a TipcMt object that specifies the type of message that this process event handles. If *message_type* is null, the event handler processes all message types.

Return Type None

C API TipcSrvSubjectCbLookup, TutCbDestroy

Example This example removes callbacks:

```
Dim s as TipcSrv, mtInfo as new TipcMt
Set s = TipcSvc1.Srv
mtInfo.Lookup("info")

'Destroy subject callback for all info messages received
s.RemoveSubjectCb TipcProcessCb1.Object, "/...", mtInfo

'Destroy subject callback for ALL messages received
s.RemoveSubjectCb TipcProcessCb2.Object, Null, Null

'Destroy subject cb for all info messages recvd by "/stocks"
s.RemoveSubjectCb TipcProcessCb3.Object, "/stocks", mtInfo
```


RemoveSubjectCbAll Method

The RemoveSubjectCbAll method uninstalls all subject callbacks that were previously installed with AddSubjectCb or AddSubjectDefaultCb.

Syntax `object.RemoveSubjectCbAll`

where *object* is an object expression that evaluates to a TipcSrv object.

Return Type None

C API TipcSrvSubjectCbDestroyAll

Example This example destroys all previously registered subject callbacks:

```
Dim s as TipcSrv
Set s = TipcSvc1.Srv
s.RemoveSubjectCbAll
```

RemoveSubjectDefaultCb Method

The RemoveSubjectDefaultCb method uninstalls default subject callbacks. These default subject callbacks are installed with the AddSubjectDefaultCb method.

Syntax `object.RemoveSubjectDefaultCb event_handler`

where:

object is an object expression that evaluates to a TipcSrv object.

event_handler evaluates to a TipcProcessCb object whose TipcMsg event is invoked when no other subject callbacks are processed.

Return Type None

C API TipcSrvSubjectDefaultCbLookup, TutCbDestroy

Example This example removes a default subject callback:

```
Dim s as TipcSrv
Set s = TipcSvc1.Srv
'Destroy default subject callback
s.RemoveSubjectDefaultCb TipcProcessCb2.Object
```

StdSubjectRetrieve Method

The StdSubjectRetrieve method retrieves the current value of the standard subjects.

Syntax `object.StdSubjectRetrieve retrieve_time`
where:
`object` is an object expression that evaluates to a TpcSrv object.
`retrieve_time` is a Boolean value that specifies whether the `_time` subject should be retrieved.

Return Type None

Remarks Standard subjects are a way to automatically subscribe to one or more subjects. To do this, set the Subjects option in the command file associated with the RTclient. For more information on standard subjects, see the reference page for TpcSrvStdSubjectRetrieve in the *TIBCO SmartSockets Application Programming Interface*.

C API TpcSrvStdSubjectRetrieve

Example This example retrieves the standard subjects for this RTclient:

```
Dim s As TpcSrv
Set s = TpcSvc1.Srv
s.StdSubjectRetrieve False
```

StdSubjectSubscribe Method

The StdSubjectSubscribe method starts or stops subscribing to the standard subjects.

Syntax `object.StdSubjectSubscribe subscribe_status, time_status`
where:
`object` is an object expression that evaluates to a TpcSrv object.
`subscribe_status` is a Boolean value. `True` starts, and `False` stops subscribing to the standard subjects.
`time_status` is a Boolean value that specifies whether the `_time` subject should be received (subscribed to).

Return Type None

Remarks	Standard subjects are a way to automatically subscribe to one or more subjects. To do this, set the Subjects option in the command file associated with the RTclient. For more information on standard subjects, see the reference page for TipcSrvStdSubjectRetrieve in the <i>TIBCO SmartSockets Application Programming Interface</i> .
C API	TipcSrvStdSubjectSubscribe
Example	This example subscribes to the standard subjects for this RTclient: <pre>Dim s As TipcSrv Set s = TipcSvc1.Srv s.StdSubjectSubscribe True, False</pre>

SubjectGmdInit Method

The SubjectGmdInit method initializes GMD accounting for the specified subject.

Syntax `object.SubjectGmdInit subject`

where:

`object` is an object expression that evaluates to a TipcSrv object.

`subject` is a string that specifies the subject for initializing GMD.

Return Type None

Remarks This method is not usually used by application developers because publishing with GMD automatically starts GMD accounting. It can, however, be used to prime the RTserver to be ready for publishing large numbers of GMD messages. For more information on GMD, see the *TIBCO SmartSockets User's Guide*.

C API TipcSrvSubjectGmdInit

Example This example initializes GMD for the "/stocks" subject:

```
Dim s As TipcSrv
Set s = TipcSvc1.Srv
s.SubjectGmdInit "/stocks"
```

SubjectLbInit Method

The SubjectLbInit method initializes load balancing accounting for the specified subject.

Syntax `object.SubjectLbInit subject`
where:
`object` is an object expression that evaluates to a TpcSrv object.
`subject` is a string that specifies the subject for initializing load balancing accounting.

Return Type None

Remarks This method is not usually used by application developers because publishing with load balancing automatically starts load balancing accounting. It can, however, be used to prime the RTserver to be ready for publishing large numbers of load balanced messages. Currently, both GMD and load balancing accounting use the same mechanism—choose the appropriate method for the desired action. For more information on load balancing, see the *TIBCO SmartSockets User's Guide*.

C API TpcSrvSubjectLbInit

Example This example initializes load balancing for the "/stocks" subject:

```
Dim s As TpcSrv
Set s = TpcSvc1.Srv
s.SubjectLbInit "/stocks"
```

SubjectRetrieve Method

The SubjectRetrieve method is used with prior versions of SmartSockets and is not available in SmartSockets Version 5 and higher.

TipcSrv Class Properties

Arch Property

The Arch property contains a string indicating the hardware architecture on which RTserver is running and is used to handle conversion to and from machines with differing data formats. This property is read-only.

Syntax *object*.Arch
 where *object* is an object expression that evaluates to a TipcSrv object.

Data Type String

C API TipcSrvGetArch

Example This example displays the architecture of RTserver:
 Debug.Print "RTserver runs "; TipcSvc1.Srv.Arch

AutoFlushSize Property

The AutoFlushSize property retrieves or sets the number of outgoing bytes buffered before being automatically flushed to an RTserver.

Syntax *object*.AutoFlushSize [=flush_size]
 where:
object is an object expression that evaluates to a TipcSrv object.
flush_size is an integer that specifies the amount of outgoing data, in bytes, to buffer before automatically flushing the connection to the RTserver. The default value is 8192.

Data Type Long

C API TipcSrvGetAutoFlushSize, TipcSrvSetAutoFlushSize

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example doubles the buffer size before autoflushing, until it is at least 16K:

```
While (TipcSvc1.Srv.AutoFlushSize < 16384)_  
    TipcSvc1.Srv.AutoFlushSize =  
        TipcSvc1.Srv.AutoFlushSize * 2  
Wend
```

BlockMode Property

The BlockMode property retrieves or sets the input/output (I/O) mode used for the connection to an RTserver. The mode is blocking or non-blocking. The default value is True.

Syntax *object*.BlockMode [=*block_mode*]
where:
object is an object expression that evaluates to a TipcSrv object.
block_mode is a Boolean value. True sets the blocking mode and False sets the non-blocking I/O.

Data Type Boolean

C API TipcSrvGetBlockMode, TipcSrvSetBlockMode

Example This example sets the connection to the RTserver for non-blocking mode:

```
Dim s As TipcSrv  
Set s = TipcSvc1.Srv  
s.BlockMode = False
```

ConnStatus Property

The ConnStatus property retrieves or sets the connection status to an RTserver.

Syntax *object*.ConnStatus [=*conn_status*]
where:
object is an object expression that evaluates to a TipcSrv object.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

conn_status is class `TipcSrvConnStatus` and defines the connection to the RTserver:

- 0 is `tipcSrvConnNone`, which is no connection to RTserver
- 1 is `tipcSrvConnWarm`, which is a warm connection to RTserver
- 2 is `tipcSrvConnFull`, which is a full connection to RTserver

Data Type	TipcSrvConnStatus
Remarks	Setting this property directly, allows your application to go from a full to a warm RTserver connection. To connect or fully disconnect from RTserver, it is often easier and more explicit to directly call the Create and Destroy methods. For more information, see the <i>TIBCO SmartSockets User's Guide</i> .
C API	TipcSrvGetConnStatus
Example	This example retrieves the status of the RTserver connection: <pre>Dim status As TipcSrvConnStatus status = TipcSvc1.Srv.ConnStatus Select Case status Case tipcSrvConnNone Debug.Print "Server connection status: None" Case tipcSrvConnWarm Debug.Print "Server connection status: Warm" Case tipcSrvConnFull Debug.Print "Server connection status: Full" End Select</pre>

GmdDir Property

The GmdDir property specifies where the file-based GMD message area is located on the local machine. This area is used to store unacknowledged GMD messages (for example, in case of power interruption) and messages are removed as they are successfully delivered to all recipients. This property is read-only and is not set unless GMD is available (that is, RTclient has a unique subject name set).

Syntax	<i>object</i> .GmdDir where <i>object</i> is an object expression that evaluates to a TipcSrv object.
Data Type	String
C API	TipcGetGmdDir

Example This example shows where the local GMD area is:

```
Debug.Print "Gmd Directory is: " & TipcSvc1.Srv.GmdDir
```

GmdMaxSize Property

The GmdMaxSize property retrieves or sets the maximum amount of disk space or memory the GMD area is allowed to consume. The default value is 0, meaning that there is no limit to the amount of space used. This property cannot be changed while there is an active connection to an RTserver.

Syntax *object*.GmdMaxSize [=*max_size*]
where:
object is an object expression that evaluates to a TipcSrv object.
max_size is an integer that specifies the maximum amount of memory or space, in bytes, that the GMD area is allowed to use. Using 0 sets no limit.

Data Type Long
C API TipcSrvGetGmdMaxSize, TipcSrvSetGmdMaxSize

Example This example retrieves the maximum size of the GMD area:

```
Debug.Print "Max GMD size: " &  
Format(TipcSvc1.Srv.GmdMaxSize)
```

GmdNumPending Property

The GmdNumPending property retrieves the number of GMD messages still pending to be sent to an RTserver. This property is read-only.

Syntax *object*.GmdNumPending
where *object* is an object expression that evaluates to a TipcSrv object.

Data Type Long
C API TipcSrvGetGmdNumPending

Example

This example retrieves the number of GMD messages pending:

```
Debug.Print "Number of pending GMD messages: " &
    Format(TipcSvc1.Srv.GmdNumPending)
```

Lcn Property

The Lcn property returns the Logical Connection Name that the RTclient used to communicate with the RTserver and is provided for reference only; it is used internally when connected to an RTserver. This property is read-only.

Syntax	<code>object.Lcn</code> where <i>object</i> is an object expression that evaluates to a TipcSrv object.
Data Type	String
Remarks	For more information on logical connection names, see the <i>TIBCO SmartSockets User's Guide</i> .
C API	TipcConnGetName
Example	This example retrieves the logical connection name of the running RTclient: <pre>Dim status As TipcSrvConnStatus Dim s As TipcSrv Set s = TipcSvc1.Srv status = s.ConnStatus If status <> tipcSrvConnNone Then Debug.Print "Connection name: " & s.lcn End If</pre>

Node Property

The Node property retrieves the node name of the connected RTserver. This property is read-only.

Syntax	<code>object.Node</code> where <i>object</i> is an object expression that evaluates to a TipcSrv object.
Data Type	String
C API	TipcSrvGetNode

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example retrieves the node name of the connected RTserver:

```
Dim status As TipcSrvConnStatus
Dim s As TipcSrv

Set s = TipcSvc1.Srv
status = s.ConnStatus
If status <> tipcSrvConnNone Then
    Debug.Print "Node name: " & s.node
End If
```

NumQueued Property

The NumQueued property retrieves the number of messages, still pending, to be read by RTclient. This property is read-only.

Syntax *object*.NumQueued
where *object* is an object expression that evaluates to a TipcSrv object.

Data Type Long

C API TipcSrvGetNumPending

Example This example retrieves the number of messages queued:

```
Dim s As TipcSrv
Dim msg as TipcMsg
Set s = TipcSvc1.Srv
If s.NumQueued > 0 Then
    set msg = s.MsgNext(0.0)
    s.MsgProcess msg
    msg.Destroy
End If
```

Pid Property

The Pid property retrieves the operating system Process Identification Number (PID) of the connected RTserver for diagnostic purposes. This property is read-only.

Syntax *object*.Pid
where *object* is an object expression that evaluates to a TipcSrv object.

Data Type Long

C API TipcSrvGetPid

Example This example retrieves the PID of the connected RTserver:

```
Dim status As TipcSrvConnStatus
Dim s As TipcSrv

Set s = TipcSvc1.Srv
status = s.ConnStatus
If status <> tipcSrvConnNone Then
    Debug.Print "RTserver pid: " & s.pid
End If
```

Subjects Property

The Subjects property retrieves the subjects to which RTclient is subscribed. This property is read-only.

Syntax *object*.Subjects

where *object* is an object expression that evaluates to a TipcSrv object.

Data Type TipcSubjects

Remarks See TipcSubjects Class Methods on page 227 for more information.

C API TipcSrvSubjectGetSubscribe, TipcMonSubjectSubscribeSetWatch

Example This example lists the subjects to which RTclient is subscribed:

```
Dim sb as TipcSubjects

Set sb = TipcSvc1.Srv.Subjects
For t% = 0 to sb.Count - 1
    Debug.Print sb(t%)
Next t%
```

Timeout Property

The Timeout property retrieves or sets the Read, Write, GMD Delivery or KeepAlive timeout values.

Syntax *object*.Timeout (*timeout_type*) [=*timeout*]

where:

object is an object expression that evaluates to a TipcSrv object.

timeout_type is class `TipcTimeout` and specifies which type of timeout to set or return:

- 0 is `tipcTimeoutRead` which sets the server read timeout
- 1 is `tipcTimeoutWrite` which sets the server write timeout
- 2 is `tipcTimeoutKeepAlive` which sets the server keep-alive timeout
- 3 is `tipcTimeoutDelivery` which sets the GMD message delivery timeout

timeout is a double real number which specifies the length of the timeout, in seconds.

Data Type	Double
Remarks	These values are also available from the parent <code>TipcSvc</code> object's properties <code>ServerReadTimeout</code> , <code>ServerWriteTimeout</code> , <code>ServerKeepAliveTimeout</code> and <code>ServerDeliveryTimeout</code> . See the sections for those properties for further information.
C API	<code>TipcSrvGetTimeout</code> , <code>TipcSrvSetTimeout</code>
Example	This example sets all the timeout values to one minute: <pre>Dim s As TipcSrv Set s = TipcSvc1.Srv s.Timeout tipcTimeoutRead = 60.0 s.Timeout tipcTimeoutWrite = 60.0 s.Timeout tipcTimeoutKeepAlive = 60.0 s.Timeout tipcTimeoutDelivery = 60.0 Debug.Print "Read: " & Format(s.Timeout(tipcTimeoutRead)) Debug.Print "Write: " & Format(s.Timeout (tipcTimeoutWrite)) Debug.Print "KA: " & Format(s.Timeout(tipcTimeoutKeepAlive)) Debug.Print "Del: " & Format(s.Timeout (tipcTimeoutDelivery))</pre>

UniqueSubject Property

The `UniqueSubject` property retrieves the unique subject name of the `RTclient`. Direct access to this property is read-only, but the value can be modified through a `SmartSockets` `setopt` command.

Syntax `object.UniqueSubject`
where *object* is an object expression that evaluates to a `TipcSrv` object.

Data Type	String
Remarks	See ParseStr Method on page 236 and UniqueSubject Property on page 252 for more information on setting the UniqueSubject property.
C API	TipcSrvGetUniqueSubject
Example	This example retrieves the unique subject for RTclient: <pre>Debug.Print "Connected to the RTserver with subject: " & Format(TipcSvc1.Srv.UniqueSubject)</pre>

User Property

The User property retrieves the user name of the connected RTserver and is useful for diagnostic purposes. This property is read-only.

Syntax	<i>object</i>.User where <i>object</i> is an object expression that evaluates to a TipcSrv object.
Data Type	String
C API	TipcSrvGetUser
Example	This example retrieves the user name of the connected RTserver: <pre>Dim status As TipcSrvConnStatus Dim s As TipcSrv Set s = TipcSvc1.Srv status = s.ConnStatus If status <> tipcSrvConnNone Then Debug.Print "User: " & s.User End If</pre>

TipcSubjects Class

TipcSubjects — user class for managing publish-subscribe participation

Description	The TipcSubjects class provides an interface to the subjects to which an RTclient is subscribed. It allows you to add, remove, retrieve and count the subjects contained within. For more information on subjects, see the <i>TIBCO SmartSockets User's Guide</i> .
Methods	• Add Method • Remove Method
Properties	• Count Property • Item Property
Events	None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcSubjects Class Methods

Add Method

The Add method subscribes to and determines participation in the load balancing calculations for the specified subject.

Syntax

`object.Add subject, lb_status`
where:
`object` is an object expression that evaluates to a TipcSubjects object.
`subject` is a string that contains the publish-subscribe subject to subscribe to.
`lb_status` is a Boolean value that specifies whether to participate in load balancing for this subject:

- `True` means the RTclient receives messages using the load balancing mode set for this subject
- `False` means that the RTclient always receives messages published to this subject

Return Type None

C API TipcSrvSubjectSetSubscribeLb

Example

This example subscribes to the two subjects, one as part of the load balancing group, and one to always receive all messages:
`TipcSvc1.Srv.Subjects.Add "/demo/sub_one",False`
`TipcSvc1.Srv.Subjects.Add "/demo/load_bal",True`

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Remove Method

The Remove method unsubscribes an RTclient from the specified subject.

Syntax *object.Remove subject*
where:
object is an object expression that evaluates to a TipcSubjects object.
subject is a string that contains the publish-subscribe subject to stop subscribing to.

Return Type None

C API TipcSrvSubjectSetSubscribeLb

Example This example unsubscribes from a subject:
`TipcSvc1.Srv.Subjects.Remove "/demo/sub_one"`

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcSubjects Class Properties

Count Property

The Count property returns the number of subjects contained in the TipcSubjects object. This is a read-only value.

Syntax *object*.Count
where *object* is an object expression that evaluates to a TipcSubjects object.

Data Type Long

C API No direct counterpart in the C API, similar function is achieved within the Monitoring API.

Example This example prints out the number of subjects to which this RTclient is subscribed:

```
Dim ct As Long

ct = TipcSvc1.Subject.Count
Debug.Print "subscribed to " & Format(ct) & " subjects"
```

Item Property

The Item property returns the subject name being subscribed to with the specified index. The first subject is specified by index 0. This property is read-only and can only be modified by the Add and Remove methods.

Syntax *object*.Item(*index*)
where:
object is an object expression that evaluates to a TipcSubjects object.
index is a long integer that specifies which subscribed subject to return. Using 0 returns the first field.

Data Type String

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Remarks The word "item" has special meaning in most ActiveX environments; the `Item` keyword can be omitted and the index in parentheses applied directly to the object. See the example below.

C API No comparable function.

Example This example uses the `Item` property, first directly and then indirectly, to display the first two subjects to which this `RTclient` is subscribed:

```
Debug.Print TipcSvc1.Srv.Subjects.Item(0)
Debug.Print TipcSvc1.Srv.Subjects(1)
```

This example prints out all of the subjects subscribed to:

```
Dim ct As Long
Dim i As Long

ct = TipcSvc1.Srv.Subjects.Count
For i = 0 to ct - 1
    Debug.Print TipcSvc1.Srv.Subjects(i)
Next i
```

TipcSvc Class

TipcSvc — user class for controlling the connection to an RTserver

Description The TipcSvc class provides an interface to the options used to connect with RTserver and SmartSockets messages. It allows you to set various timeouts, to specify to which RTservers to connect, and to create new SmartSockets messages. For more information on RTserver, see the *TIBCO SmartSockets User's Guide*.

Methods

- AboutBox Method
- CheckVersion Method
- MsgCreate Method
- MtCreate Method
- MtLookup Method
- ParseFile Method
- ParseStdLicense Method
- ParseStr Method
- XmlCreate Method

Properties

- AutoMainLoop Property
- DefaultMsgPriority Property
- DefaultSubjectPrefix Property
- EnableControlMsgs Property
- InputEventsStrategy Property
- LastErrorNum Property
- LastErrorStr Property
- MonitorScope Property
- Option Property
- Project Property
- ServerAutoConnect Property
- ServerAutoFlushSize Property
- ServerDeliveryTimeout Property
- ServerKeepAliveTimeout Property
- ServerMsgSend Property
- ServerNames Property
- ServerReadTimeout Property
- ServerStartDelay Property
- ServerStartMaxTries Property
- ServerStartTimeout Property
- ServerWriteTimeout Property
- SocketConnectTimeout Property
- Srv Property
- Subjects Property
- UdpBroadcastTimeout Property
- UniqueSubject Property

Events

- DataArrival Event
- SrvCreate Event
- SrvDestroy Event

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcSvc Class Methods

AboutBox Method

The AboutBox method displays the modal AboutBox for the SmartSockets TipcSvc ActiveX control, which includes version and copyright information.

Syntax *object*.AboutBox
where *object* is an object expression that evaluates to a TipcSvc object.

Return Type None

C API No comparable C API function

Example This example pops up the About box:
`TipcSvc1.AboutBox`

CheckVersion Method

The CheckVersion method checks the version of the SmartSockets ActiveX control.

Syntax *object*.CheckVersion *major*, *minor*
where:
object is an object expression that evaluates to a TipcSvc object.
major is an integer that specifies the major number of the release.
minor is an integer that specifies the minor number of the release.

Return Type Boolean

C API None

Example This example checks the version of the TpcSvc SmartSockets control:

```
Dim bBool as Boolean
bBool = TpcSvc1.CheckVersion(2, 0)
if(bBool = False) then
    MsgBox("Invalid Version")
End If
```

MsgCreate Method

The MsgCreate method is similar to the TpcMsg.Create method. It creates a message object containing a SmartSockets message of the specified type.

Syntax `object.MsgCreate mt`

where:

`object` is an object expression that evaluates to a TpcSvc object.

`mt` is an initialized TpcMt object.

Return Type TpcMsg

Remarks It is often easier to use this method to create new messages than to call the lower-level methods of TpcMsg directly. For more information, see the *TIBCO SmartSockets User's Guide*.

C API TpcMsgCreate

Example This example puts a new info message in the msg object:

```
Dim mt as New TpcMt, msg as TpcMsg
mt.Lookup("info")
Set msg = TpcSvc1.MsgCreate(mt)
```

MtCreate Method

The MtCreate method creates a new message type object containing a SmartSockets message of the specified type.

Syntax `object.MtCreate (name, num, grammar)`

where:

`object` is an object expression that evaluates to a TpcSvc object.

name is a string representing the name of the message type. It is not case-sensitive.

num is a long integer that specifies the numerical ID being assigned to the new message type.

grammar is a string that specifies the description of the message's format used for reading and writing message files.

Return Type	TipcMt
Remarks	See the <i>TIBCO SmartSockets User's Guide</i> for more information about grammar and field types, as well as numeric message IDs. It is often easier to use this method to create new message types than to call the lower-level methods of TipcMt directly.
C API	TipcMtCreate
Example	This example creates a new message type <i>two_strings</i> with numeric ID 2003, and a grammar of two string fields: <pre>Dim mt as TipcMt Set mt = TipcSvc1.MtCreate("two_strings", 2003, "str str")</pre>

MtLookup Method

The MtLookup method creates a new message type object containing a SmartSockets message of the specified pre-existing type, indicated by the numeric ID or descriptive name provided.

Syntax *object*.MtLookup (*name_or_num*)

where:

object is an object expression that evaluates to a TipcSvc object.

name_or_num is a string or a long integer. If you specify a name, it is string representing the name of the message type. It is not case-sensitive. If you specify a number, it is a long integer representing the assigned numeric ID of the message type.

Return Type	TipcMt
C API	TipcMtLookup

Example This example shows two different ways to initialize a TipcMt object to the numeric_data message type:

```
Dim mt as TipcMt
Set mt = TipcSvc1.MtLookup("numeric_data")
Set mt = TipcSvc1.MtLookup(-22)
```

ParseFile Method

The ParseFile method parses a SmartSockets command file. ParseFile returns True if the file was accessible, False if it was unable to access the file.

Syntax `object.ParseFile (filename)`

where:

object is an object expression that evaluates to a TipcSvc object.

filename is a string that specifies the name of the command file to parse, including the drive and pathname if needed.

Return Type Boolean, where:
 True means the file was accessible.
 False means the file could not be accessed.

Remarks A True result does not necessarily imply the operations specified in the file were completed successfully. For more information on command files, see the *TIBCO SmartSockets User's Guide*.

C API TutCommandParseFile

Example This example illustrates how to use ParseFile to read in a SmartSockets command file located on the C drive:

```
Dim success as Boolean
success = TipcSvc1.ParseFile("c:\demos\init.cm")
```

ParseStdLicense Method

The ParseStdLicense method loads the system-level license command file `license.cm`.

Syntax `object.ParseStdLicense`

where *object* is an object expression that evaluates to a TipcSvc object.

Return Type Boolean

C API None

Example This example parses the system-level license command file:

```
Dim bBool as Boolean
bBool = TipcSvc1.ParseStdLicense
if(bBool = False) then
    MsgBox("Unable to parse license.cm")
End If
```

ParseStr Method

The ParseStr method parses a SmartSockets command string.

Syntax *object.ParseStr command_string*
where:
object is an object expression that evaluates to a TipcSvc object.
command_string is a string that specifies the SmartSockets command to parse.

Return Type None

Remarks For more information on valid commands, see the *TIBCO SmartSockets User's Guide*.

C API TutCommandParseStr

Example This example uses the ParseStr method to set the RTclient's project:
TipcSvc1.ParseStr("setopt project activex_demo")

XmlCreate Method

The XmlCreate method is similar to the TipcXml.Create method. It creates an XML object containing a string of XML data.

Syntax *object.XmlCreate xml_data*

where:

object is an object expression that evaluates to a TipcSvc object.

xml_data is a string that specifies the XML data the returned TipcXml object contains.

Return Type TipcXml

C API TutXmlCreate

Example This example creates a TipcXml object using TipcSvc:

```
Dim tXml As TipcXml
Set tXml =
TipcSvc1.XmlCreate("<data><data1><temp=hot/></data1></data>")
```

TipcSvc Class Properties

AutoMainLoop Property

The AutoMainLoop property specifies whether or not DataArrival events should be handled by automatically calling the TipcSrv.MainLoop method.

Syntax *object*.AutoMainLoop [=auto_main_loop]
where:
object is an object expression that evaluates to a TipcSvc object.
auto_main_loop is a Boolean value that specifies the automatic processing mode:

- True automatically calls TipcSrv.MainLoop when data arrives to RTclient for processing. The DataArrival event is never invoked.
- False invokes the DataArrival event when data is available to RTclient. RTclient is responsible for calling the MainLoop method. This is the default.

Data Type Boolean

Remarks AutoMainLoop invokes the MainLoop method, so using AutoMainLoop means not having to call MainLoop directly.

C API No comparable C API function

Example This example enables the MainLoop method to process messages as they are received:

TipcSvc1.AutoMainLoop = True

DefaultMsgPriority Property

The DefaultMsgPriority property specifies the default priority of the messages sent through this connection. This priority can be overridden by setting the priority on individual messages or by using message types with different priorities.

Syntax `object.DefaultMsgPriority[=priority]`

where:

object is an object expression that evaluates to a TipcSvc object.

priority is an integer that specifies the default priority for messages sent on this connection.

Data Type Integer

C API Default_Msg_Priority option

Example This example sets the default priority of all messages sent out with this connection to a value of 10:

```
TipcSvc1.DefaultMsgPriority = 10
```

DefaultSubjectPrefix Property

The DefaultSubjectPrefix property specifies the default text prepended to subject names used by RTclient that do not begin with a slash (/).

Syntax `object.DefaultSubjectPrefix [=prefix]`

where:

object is an object expression that evaluates to a TipcSvc object.

prefix is a string that specifies the text to be prefixed to subjects that are not fully qualified, that is, that do not begin with a slash (/).

Data Type String

Remarks See the *TIBCO SmartSockets User's Guide* for detailed information about subject prefixes.

C API Default_Subject_Prefix option

Example This example adds "/StockTicker" to the beginning of each subject name that is not fully qualified:

```
TipcSvc1.DefaultSubjectPrefix = "/StockTicker"
```

EnableControlMsgs Property

The EnableControlMsgs property specifies the commands allowed in a control message. See the *TIBCO SmartSockets User's Guide* for more information on control message enabling. The default value is "echo, quit."

Syntax `object.EnableControlMsgs [=commands]`
where:
`object` is an object expression that evaluates to a TipcSvc object.
`commands` is a string that specifies the commands allowed in a control message. The commands in the string must be separated by commas.

Data Type String

C API Enable_Control_Msgs option

Example This example shows the control messages enabled:

```
Dim s as String
s = TipcSvc1.EnableControlMsgs
If (s <> "echo, quit") Then
    MsgBox "Possible security hole!", 0, "Security
        Warning!"
End If
```

InputEventsStrategy Property

The InputEventsStrategy property specifies the setting of the event strategy for processing messages when AutoMainLoop is True.

Syntax `object.InputEventsStrategy [=strategy]`
where:
`object` is an object expression that evaluates to a TipcSvc object.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

strategy is an integer that specifies the strategy for receiving messages from RTserver:

- 0 is `tipcEventsMaxSpeed`, which provides higher service for receiving messages. However, during processing, less service is given to the user interface. Use this option to process a high volume of message traffic in a short period of time
- 1 is `tipcEventsResponsiveGUI`, which provides higher service to the GUI. However, messages are not processed as quickly.

C API No comparable C API function

Example This example sets the `InputEventsStrategy` so that the user interface is given more time to process user events:

```
TipcSvc1.InputEventsStrategy = tipcEventsResponsiveGUI
```

LastErrorNum Property

The `LastErrorNum` property returns the global SmartSockets error number. This is a read-only value.

Syntax `object.LastErrorNum`
where *object* is an object expression that evaluates to a `TipcSvc` object.

Return Type Long

C API `TutErrNumGet`

Example This example returns the last SmartSockets error number:

```
Dim error_num As Long
error_num = Tipc.LastErrorNum
```

LastErrorStr Property

The `LastErrorStr` property returns the global SmartSockets error number as a descriptive string. This is a read-only value.

Syntax `object.LastErrorStr`

where *object* is an object expression that evaluates to a TipcSvc object.

Return Type String

C API TutErrStrGet

Example This example returns the last SmartSockets error string:

```
Dim error_str As String
error_str = Tipc.LastErrorStr
```

MonitorScope Property

The MonitorScope property specifies the level of interest for SmartSockets monitoring in those monitoring categories with no parameters (for example, RTclient names poll) or a parameter of @ (for example, RTclient time watch). MonitorScope acts as a filter, allowing a subset of the subject namespace to be specified. This is useful for RTclients monitoring only a subset of a large project.

Syntax *object*.MonitorScope [=*monitor_scope*]
where:
object is an object expression that evaluates to a TipcSvc object.
monitor_scope is a string that specifies the monitoring scope of the RTclient.

Data Type String

Remarks For more information on monitoring scope, see the *TIBCO SmartSockets User's Guide*.

C API Monitor_Scope option

Example This example monitors only the cars subjects in the subject namespace:

```
TipcSvc1.MonitorScope = "/stocks/cars/..."
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Option Property

The Option property retrieves or sets a specified option.

Syntax `object.Option option [=value]`
where:
object is an object expression that evaluates to a TipcSvc object.
option is a string that specifies the name of the option to set or get.
value is a variant that specifies what to set the option to. If you do not specify a value, the Option property retrieves the value of the option.

Return Type Variant

C API TutOptionGet*, TutOptionSet*

Example This example sets and retrieves options:

```
Dim Unique_Subject As String
Unique_subject = TipcSvc1.Option("unique_subject")
MsgBox (Unique_Subject)

Tipc.Option("server_delivery_timeout") = 10
```

Project Property

The Project property specifies to which project the RTclient connects.

Syntax `object.Project [=project]`
where:
object is an object expression that evaluates to a TipcSvc object.
project is a string that specifies the name of the project to which RTclient connects. The default value is `rtworks`. This value cannot be changed while a connection to the RTserver is active.

Data Type String

Remarks The Project option can only be set prior to the connection with the RTserver. For more information, see the *TIBCO SmartSockets User's Guide*.

C API Project option

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example sets the project to "tibco":

```
TipcSvc1.Project = "tibco"
Debug.Print "Connecting with project " & TipcSvc1.Project
```

ServerAutoConnect Property

The ServerAutoConnect property specifies whether or not the ActiveX controls should automatically try to create a connection to an RTserver.

Syntax `object.ServerAutoConnect [=auto_connect]`
where:
`object` is an object expression that evaluates to a TipcSvc object.
`auto_connect` is a Boolean value where:

- True allows the ActiveX components to automatically connect to the RTserver. This is the default value.
- False does not allow the ActiveX components to automatically connect to the RTserver.

Data Type Boolean

C API Server_Auto_Connect option

Example This example disables the ability to automatically connect to the RTserver:

```
TipcSvc1.ServerAutoConnect = False
```

ServerAutoFlushSize Property

The ServerAutoFlushSize property specifies the number of bytes buffered before data is automatically sent to an RTserver.

Syntax `object.ServerAutoFlushSize [=flush_size]`
where:
`object` is an object expression that evaluates to a TipcSvc object.
`flush_size` is a long integer that specifies the number of bytes to buffer before flushing data to an RTserver.

Data Type	Long
C API	Server_Auto_Flush_Size option
Example	This example prints out the value of this property: <pre>Debug.Print "AutoFlushSize value is: " & Format(TipcSvc1.ServerAutoFlushSize)</pre>

ServerDeliveryTimeout Property

The ServerDeliveryTimeout property specifies the default maximum interval, in seconds, allowed for all receiving processes to acknowledge a message sent with GMD.

Syntax	<i>object</i> .ServerDeliveryTimeout [= <i>timeout</i>] where: <i>object</i> is an object expression that evaluates to a TipcSvc object. <i>timeout</i> is a double real number that specifies the maximum GMD acknowledgement interval in seconds. The default value is 30.0.

Data Type	Double
C API	Server_Delivery_Timeout option
Example	This example sets the server delivery timeout to 20 seconds: <pre>TipcSvc1.ServerDeliveryTimeout = 20 If TipcSvc1.ServerDeliveryTimeout <> 20 Then Debug.Print "Unable to assign ServerDeliveryTimeout" End If</pre>

ServerKeepAliveTimeout Property

The ServerKeepAliveTimeout property specifies the maximum amount of time to wait, in seconds, when checking to see if the connection to an RTserver is still alive.

Syntax	<i>object</i> .ServerKeepAliveTimeout [= <i>timeout</i>] where: <i>object</i> is an object expression that evaluates to a TipcSvc object.

timeout is a double real number that specifies the maximum time in seconds to wait to determine if the RTserver is still alive. The default value is 5.0.

Data Type	Double
C API	Server_Keep_Alive_Timeout option
Example	This example sets the server keep alive timeout to 20 seconds: <pre>TipcSvc1.ServerKeepAliveTimeout = 20 If TipcSvc1.ServerKeepAliveTimeout <> 20 Then Debug.Print "Unable to assign ServerKeepAliveTimeout" End If</pre>

ServerMsgSend Property

The ServerMsgSend property specifies whether or not RTclient is allowed to send messages to an RTserver, or if messages are discarded without transmission. This is useful for backup processes that receive, but do not send messages.

Syntax	<i>object</i>.ServerMsgSend [=enable_send] where: <i>object</i> is an object expression that evaluates to a TipcSvc object. <i>enable_send</i> is a Boolean value: • True allows the RTclient to transmit messages to the RTserver. This is the default value. • False does not allow the RTclient to send messages to the RTserver.

Data Type	Boolean
C API	Server_Msg_Send option
Example	This example will not transmit any message with the MsgSendEx property: <pre>TipcSvc1.ServerMsgSend = False TipcSvc1.Srv.MsgSend msg 'Msg will not be sent TipcSvc1.Srv.MsgSendEx msg, True 'Msg will not be sent TipcSvc1.Srv.MsgSendEx msg, False ' Msg will be sent TipcSvc1.Srv.Flush</pre>

ServerNames Property

The ServerNames property specifies a list of logical connection names used to find and start RTserver.

Syntax `object.ServerNames [=server_names]`
where:
`object` is an object expression that evaluates to a TipcSvc object.
`server_names` is a string that specifies the logical connection names used to find and start an RTserver. The default value is `_node` which specifies the local machine.

Data Type String

Remarks Logical connection names have the form *protocol:node:address*, which can be shortened to *protocol:node*, *protocol*, or *node*. For more information, see the *TIBCO SmartSockets User's Guide*.

C API Server_Names option

Example This example connects to a remote RTserver before failing over to the local RTserver:
`TipcSvc1.ServerNames = "tcp:workstation1, _node"`

ServerReadTimeout Property

The ServerReadTimeout property specifies how often, in seconds, data is expected to be available for reading on the connection to an RTserver. This setting is used to check for possible network failures.

Syntax `object.ServerReadTimeout [=timeout]`
where:
`object` is an object expression that evaluates to a TipcSvc object.
`timeout` is a double real number that specifies how often data is expected to be available for reading, in seconds. The default value is 30.0.

Data Type Double

C API Server_Read_Timeout option

Example This example sets the server read timeout to 20 seconds:

```
TipcSvc1.ServerReadTimeout = 20
If TipcSvc1.ServerReadTimeout <> 20 Then
    Debug.Print "Unable to assign ServerReadTimeout"
End If
```

ServerStartDelay Property

The ServerStartDelay property specifies the length of a delay, in seconds, between trying the different entries in the ServerNames property (when attempting to find or start an RTserver).

Syntax *object*.ServerStartDelay [=server_start_delay]
where:
object is an object expression that evaluates to a TipcSvc object.
server_start_delay is a double real number that specifies the delay, in seconds, between trying the values in the ServerNames property. The default value is 1.0.

Data Type Double

C API Server_Start_Delay option

Example This example sets the delay to 10 seconds, if not already set:

```
If TipcSvc1.ServerStartDelay <> 10 Then
    TipcSvc1.ServerStartDelay = 10
End If
```

ServerStartMaxTries Property

The ServerStartMaxTries property specifies how many times to traverse each connection, as specified by the entries in the ServerNames property.

Syntax *object*.ServerStartMaxTries [=start_max_tries]
where:
object is an object expression that evaluates to a TipcSvc object.
start_max_tries is an integer that specifies the number of times to try connecting to an RTserver using the values in the ServerNames property. The default is 1.

Data Type	Integer
C API	Server_Start_Max_Tries option
Example	This example allows for three traversals of the ServerNames property list: <code>TipcSvc1.ServertStartMaxTries = 3</code>

ServerStartTimeout Property

The ServerStartTimeout property specifies the maximum time, in seconds, for the ActiveX components to wait for an RTserver to finish initializing.

Syntax	<code>object.ServerStartTimeout [=timeout]</code> where: <code>object</code> is an object expression that evaluates to a TipcSvc object. <code>timeout</code> is a double real number that specifies the delay in seconds between trying the values in the ServerNames property. The default value is 30.0.
Data Type	Double
C API	Server_Start_Timeout option
Example	This example sets the server start timeout to 20 seconds: <code>TipcSvc1.ServerStartTimeout = 20 If TipcSvc1.ServerStartTimeout <> 20 Then Debug.Print "Unable to assign ServerStartTimeout" End If</code>

ServerWriteTimeout Property

The ServerWriteTimeout property specifies how long, in seconds, to block the ActiveX controls from writing data to an RTserver (before initiating failure recovery). If this timeout elapses between reads, a keep-alive is sent to the RTserver.

Syntax	<code>object.ServerWriteTimeout [=timeout]</code> where: <code>object</code> is an object expression that evaluates to a TipcSvc object.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

timeout is a double real number that specifies how long to block, in seconds, while writing to the RTserver. The default value is 30.0.

Data Type	Double
C API	Server_Write_Timeout option
Example	This example sets the server write timeout to 20 seconds: <pre>TipcSvc1.ServerWriteTimeout = 20 If TipcSvc1.ServerWriteTimeout <> 20 Then Debug.Print "Unable to assign ServerWriteTimeout" End If</pre>

SocketConnectTimeout Property

The SocketConnectTimeout property specifies the maximum time, in seconds, to delay the ActiveX controls when trying to create an RTclient socket connected to an RTserver.

Syntax	<i>object</i>.ServerConnectTimeout [=<i>timeout</i>] where: <i>object</i> is an object expression that evaluates to a TipcSvc object. <i>timeout</i> is a double real number that specifies the maximum time, in seconds, to wait when attempting to create RTclient's socket to the RTserver. If set to 0.0, no timeout is enforced. The default value is 5.0.
Data Type	Double
C API	Server_Connect_Timeout option
Example	This example sets the server connect timeout to 20 seconds: <pre>TipcSvc1.SocketConnectTimeout = 20 If TipcSvc1.SocketConnectTimeout <> 20 Then Debug.Print "Unable to assign SocketConnectTimeout" End If</pre>

Srv Property

The Srv property contains the TipcSrv class object for this connection to the RTserver process. With this property, you can send messages and subscribe to subjects. Consult the reference for the TipcSrv class for more information on the capabilities of this object. This property is read-only.

Syntax *object*.Srv

where:

object is an object expression that evaluates to a TipcSvc object.

Srv is class TipcSrv and provides the RTserver interface object for the current connection.

Data Type TipcSrv

C API No comparable C API function.

Example This example uses the Srv property to create a connection to the RTserver:

```
Dim s as TipcSrv
Set s = TipcSvc1.Srv
s.Create ' create a connection to the RTserver
```

Subjects Property

The Subjects property specifies a list of publish-subscribe subjects that are added to those used when the TipcSrv.StdSubjectSubscribe method is invoked.

Syntax *object*.Subjects [=subjects]

where:

object is an object expression that evaluates to a TipcSvc object.

subjects is a string that specifies the additional subjects to include when the TipcSrv.StdSubjectSubscribe method is called.

Data Type String

Remarks Multiple values need to be separated by commas.

C API Subjects option

Example This example subscribes to these subjects when the standard subjects are subscribed:

```
TipcSvc1.Subjects = "/Stocks", "/Brokers", "/etc"  
TipcSvc1.Srv.StdSubjectSubscribe True, False
```

UdpBroadcastTimeout Property

The UdpBroadcastTimeout property specifies the maximum time, in seconds, the ActiveX controls wait when broadcasting to find an RTserver on the network.

Syntax *object*.UdpBroadcastTimeout[=*timeout*]
where:
object is an object expression that evaluates to a TipcSvc object.
timeout is a double real number that specifies the maximum time, in seconds, to wait for an RTserver to respond to broadcast requests. The default value is 5.0.

Data Type Double

C API Udp_Broadcast_Timeout option

Example This example sets the UDP broadcast timeout property to 20 seconds:

```
TipcSvc1.UdpBroadcastTimeout = 20  
If TipcSvc1.UdpBroadcastTimeout <> 20 Then  
    Debug.Print "Unable to assign UdpBroadcastTimeout"  
End If
```

UniqueSubject Property

The UniqueSubject property specifies a unique name, corresponding to the RTclient and is used when connecting to an RTserver.

Syntax *object*.UniqueSubject [=*unique_subject*]
where:
object is an object expression that evaluates to a TipcSvc object.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

unique_subject is a string that specifies the unique subject name for this RTclient. The default value is *_node_pid* where *node* is the network node name of the computer on which the ActiveX controls are running, and *pid* is the operating system process identifier of the process.

Data Type	String
Remarks	RTclient automatically subscribes to its own UniqueSubject subject. This property must be set prior to the connection to the RTserver being established.
C API	Unique_Subject option
Example	This example sets the unique subject to "MyClient": <code>TipcSvc1.UniqueSubject = "MyClient"</code>

TipcSvc Class Events

DataArrival Event

The DataArrival event is invoked when SmartSockets messages arrive. To complete processing of the data, invoke the TipcSrv.MainLoop method.

Syntax `Private Sub object_DataArrival (conn)`
 where:
 object is an object expression that evaluates to a TipcSvc object.
 conn is class TipcConn and specifies the connection on which the data arrived.

Return Type None

Remarks See the TipcSvc.MainLoop method for more information.

C API Similar to the callbacks installed with TipcConnQueueCbCreate

Example This example in a DataArrival event allows the ready-message queue to be processed:
`TipcSvc1.Srv.MainLoop 0 ' for RTserver connections`

SrvCreate Event

The SrvCreate event is invoked when the ActiveX control makes a warm or full connection to an RTserver.

Syntax `Private Sub object_SrvCreate (conn, old_conn_status, new_conn_status)`
 where:
 object is an object expression that evaluates to a TipcSvc object.
 conn is class TipcConn and specifies the connection on which RTserver was contacted.

old_conn_status is class `TipcSrvConnStatus` and indicates the status for the previous connection to RTserver:

- 0 is `tipcSrvConnNone`, which is no connection to RTserver
- 1 is `tipcSrvConnWarm`, which is a warm connection to RTserver
- 2 is `tipcSrvConnFull`, which is a full connection to RTserver

new_conn_status is class `TipcSrvConnStatus` and indicates the status for the current connection to RTserver:

- 0 is `tipcSrvConnNone`, which is no connection to RTserver
- 1 is `tipcSrvConnWarm`, which is a warm connection to RTserver
- 2 is `tipcSrvConnFull`, which is a full connection to RTserver

Return Type None

C API `TipcSrvCreateCbCreate`

Example This example creates callbacks:

```
Dim mt As New TipcMt
Dim s As TipcSrv

Set s = TipcSvc1.Srv
if old_conn_status = tipcSrvConnNone Then
    'Create Error callback
    s.AddErrorCb TipcErrorCb1.Object, 1

    'Subscribe to /stocks subject with no load balancing
    s.Subjects.Add "/stocks", False

    'Subscribe to all standard subjects
    s.StdSubjectSubscribe True, False

    'Create process callback for simple_msg message type
    mt.Lookup "simple_msg"
    s.AddProcessCb TipcProcessCb1.Object, mt

    'Create subject callback for all message types received
    'by the /stocks subject
    s.Srv.AddSubjectCb TipcProcessCb2.Object, "/stocks", Null
End If
```

SrvDestroy Event

The SrvDestroy event is invoked when the ActiveX control becomes disconnected from an RTserver.

Syntax

```
Private Sub object_SrvDestroy (conn, old_conn_status, new_conn_status)
```

where:

object is an object expression that evaluates to a TipcSvc object.

conn is class TipcConn and specifies the connection to the RTserver that was terminated.

old_conn_status is class TipcSrvConnStatus and indicates the status for the previous connection to RTserver:

- 0 is tipcSrvConnNone, which is no connection to RTserver
- 1 is tipcSrvConnWarm, which is a warm connection to RTserver
- 2 is tipcSrvConnFull, which is a full connection to RTserver

new_conn_status is class TipcSrvConnStatus and indicates the status for the current connection to RTserver:

- 0 is tipcSrvConnNone, which is no connection to RTserver
- 1 is tipcSrvConnWarm, which is a warm connection to RTserver
- 2 is tipcSrvConnFull, which is a full connection to RTserver

Return Type None

C API TipcSrvDestroyCbCreate

Example

This example destroys a connection to the RTserver:

```
If new_conn_status <> tipcSrvConnNone Then
    TipcSvc1.Srv.Destroy
End If
```

TipcXml Class

TipcXml — user class for managing XML objects.

Description The TipcXml class provides an interface to manage XML strings.

- Methods**
- Create Method
 - Clone Method
 - Destroy Method

Properties and Events None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcXml Class Methods

Create Method

The Create method defines a new SmartSockets XML object.

Syntax *object*.Create *str_data*
 where:
 object is an object expression that evaluates to a TipcXml object.
 str_data is the value of the XML string that this object will contain.

Return Type TipcXml

C API TutXmlCreate

Example This example sets the value of an XML object:

```
Dim tXml As New TipcXml
tXml.Create("<data><data1><temp=hot/></data1></data>")
```

Clone Method

The Clone method makes a copy of an existing XML object. The copied XML object does not share any memory with the original XML object.

Syntax *object*.Clone
 where *object* is an object expression that evaluates to a TipcXml object.

Return Type TipcXml

C API TutXmlClone

Example This example clones an XML object:

```
Dim tXml As New TipcXml
Dim tXml_copy As New TipcXml
tXml.Create("<data><data1><temp=hot/></data1></data>")
Set tXml_copy = tXml.Clone
```

Destroy Method

The Destroy method destroys a TipcXml object.

Syntax *object*.Destroy

where *object* is an object expression that evaluates to a TipcXml object.

Return Type TipcXml

C API TutXmlDestroy

Example This example destroys a TipcXml object:

```
Dim tXml As New TipcXml
Dim tXml_copy As New TipcXml
tXml.Create("<data><data1><temp=hot/></data1></data>")
Set tXml_copy = tXml.Clone
tXml_copy.Destroy
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Chapter 5

COM Library Reference

TIBCO SmartSockets components provide several classes, or interfaces, each with a large set of methods (callable functions), properties (settings) and events (callbacks). This chapter serves as a reference to these interfaces.

Topics

- *Relation to the TIBCO SmartSockets C API, page 262*
- *TIBCO SmartSockets COM Classes, page 262*
- *COM Error Handling, page 567*

Relation to the TIBCO SmartSockets C API

Both the COM interface and the C API upon which the COM interface is built, allow you to send and receive messages asynchronously across a heterogeneous network. The COM classes offer a method counterpart for most of the SmartSockets C API calls.

Some of the tasks that can be performed using the COM classes are:

- execute user-defined callback functions when certain operations occur
- send messages with subjects to multiple processes using the RTserver
- detect network failures and initiate actions to recover
- send messages using different IPC protocols over a heterogeneous network of computers
- monitor the application with poll and watch capabilities

TIBCO SmartSockets COM Classes

COM SmartSockets messaging applications (RTclients) connect with a client-server method, where an RTserver deals directly with your application. This method provides the greatest function and is the standard way to develop typical SmartSockets RTclient applications. The SmartSockets COM Library provides full RTserver interaction using the TipcSvc and TipcSrvConn classes. The SmartSockets COM classes and their usage are:

TipcConnChangeCb Class	is used to derive a user class for connection callbacks.
TipcDispatcher Class	provides an interface to dispatch messages received on connections and various events.
TipcErrorCb Class	provides error-handling events to RTclient applications.
TipcEvent Class	is a container class for handling dispatcher events.
TipcEventCb Class	is used to derive a user class for dispatcher events.
TipcEventMsgCb Class	is a class for events in the TipcDispatcher Class that handle messages.
TipcMsg Class	is used for constructing, manipulating, and destroying SmartSockets messages.

TipcMsgFields Class	provides access to the distinct data fields of a SmartSockets message.
TipcMt Class	is used for constructing, manipulating, looking up and destroying SmartSockets message types.
TipcProcessCb Class	handles messages arriving to your program from an event.
TipcSrvConn Class	manages a connection to RTserver and messaging tasks your application performs once connected to an RTserver.
TipcSrvConnCache Class	is used for caching data into the SmartSockets Cache process.
TipcSrvMon Class	provides monitoring for a SmartSockets application.
TipcSrvMonExt Class	is used to define, set, or delete fields for RTclient extension data.
TipcSubjects Class	is used for managing the list of publish-subscribe subjects.
TipcSvc Class	provides an interface for creating objects and enables options to be set.
TipcXml Class	is a user class for managing XML objects.
TMsgFile Class	provides an interface for creating and reading message files

TipcConnChangeCb Class

TipcConnChangeCb — class used to derive a user class for connection callbacks.

Description	The TipcConnChangeCb class provides an interface to derive user callbacks that are invoked when connections are opened or closed. The ConnChange function must be overridden by the RTclient.
Methods	None
Properties	None
Events	• ConnChange Event

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcConnChangeCb Events

ConnChange Event

The ConnChange event is triggered when a connection is opened or closed, such as in the TipcSrvConn Open Method and the TipcSrvConn Close Method.

Syntax `TipcConnChangeCb_ConnChange(
 ByVal conn As Object,
 ByVal old_conn_status As Long,
 ByVal new_conn_status As Long,
 ByVal arg As Variant)`

where:

conn is reserved.

old_conn_status is a 4-byte integer representing the connection status before this callback is invoked.

new_conn_status is a 4-byte integer representing the current connection status.

arg is a user-defined variant, which is assigned through the TipcSrvConn:: OpenCbCreate and CloseCbCreate methods.

Return Type None

C API No comparable C API function

Example This example demonstrates how to setup a callback object used in the TipcSrvConn Open Method or the TipcSrvConn Close Method:

In a separate class module, for example, GenericConnChangeCb

Implements TipcConnChangeCb

```
Private Sub TipcConnChangeCb_ConnChange(ByVal conn As Object, _
                                         ByVal old_conn_status As Long, _
                                         ByVal new_conn_status As Long, _
                                         ByVal arg As Variant)
Debug.Print "      ConnChangeCb:  old_conn_status = " & _
    + Str(old_conn_status) & " _
    + ", new_conn_status = " & _
    + Str(new_conn_status)
End Sub
```

Now, This callback object would be used as follows:

```
Dim SrvConn as TipcSrvConn
Dim Svc as New TipcSvc
Dim Conn_cb as New GenericConnChangeCallback

Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
```

```
SrvConn.OpenCbCreate Conn_cb, 1, Null
' Opening the connection will invoke the ConnChange Sub
' inside GenericConnCb
SrvConn.Open

' Now we've re-used the callback as a close callback.
' One could create a separate ClassModule with different
' behavior and register that instead
SrvConn.CloseCbCreate Conn_cb, 1, Null

' now the close will invoke the ConnChange Sub
' inside GenericConnCb
SrvConn.Close
```

TipcDispatcher Class

TipcDispatcher — class that provides an interface to dispatch messages received on connections and various events.

Description	The TipcDispatcher class dispatches messages received on connections and events. Multiple instances of TipcDispatcher objects are allowed within a process, and any number of connections and events may be added to a dispatcher. The TipcDispatcher class supports four types of events: timers, socket availability, user events and message events. Timers are invoked at a specific interval, and are repeating. Socket events are invoked for read or write availability on a socket. When a user event is added to a dispatcher, it is immediately queued to be invoked. There is not an actual trigger for user events. Message events are triggered when a message of a particular type or published to a particular subject is received on a connection.
Methods	• Create Method • CreateDetached Method • Destroy Method • Dispatch Method • MainLoop Method
Properties	None
Events	None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcDispatcher Methods

Create Method

See the TipcDispatcherCreate Method in the TipcSvc class.

CreateDetached Method

See the TipcDispatcherCreateDetached Method in the TipcSvc class.

Destroy Method

The Destroy method destroys a dispatcher.

Syntax *object*.Destroy
where *object* is an object expression that evaluates to a TipcDispatcher object.

Return Type None

C API TipcDispatcherDestroy

Example This example creates then destroys a dispatcher:

```
Dim dispatcher as TipcDispatcher
Dim svc as New TipcSvc

Set dispatcher = svc.TipcDispatcherCreate
dispatcher.Destroy
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Dispatch Method

The Dispatch method dispatches queued events.

Syntax	<i>object.Dispatch timeout</i> where: <i>object</i> is an object expression that evaluates to a TipcDispatcher object. <i>timeout</i> is a Double representing the amount of time to block for events to be queued.
Return Type	None
C API	TipcDispatcherDispatch
Example	This example creates a TipcDispatcher and Dispatch queued events for 10 seconds: <pre>Dim dispatcher as TipcDispatcher Dim svc as New TipcSvc Set dispatcher = svc.TipcDispatcherCreate ... Dispatcher.Dispatch 10 dispatcher.Destroy</pre>

MainLoop Method

The MainLoop method dispatches events.

Syntax	<i>object.MainLoop timeout</i> where: <i>object</i> is an object expression that evaluates to a TipcDispatcher object. <i>timeout</i> is a Double representing the amount of time to dispatch events for.
Return Type	None
C API	TipcDispatcherMainLoop

Example This example creates a dispatcher and waits for 10 seconds:

```
Dim dispatcher as TipcDispatcher
Dim svc as New TipcSvc
Set dispatcher = svc.TipcDispatcherCreate
' add events here
...
Dispatcher.Mainloop 10
dispatcher.Destroy
```


Remarks Typically, only one of the four values passed to the event is a valid error code. All of the parameters are passed to the event by value. You must register the TipcErrorCb event with the TipcSrvConn ErrorCbCreate Method for the TipcError event to be invoked.

C API TutErrNumGetC, TutErrNumGetOs, TutErrNumGetSocket, TutErrNumGet; also similar to user-defined functions called after registration with TipcSrvErrorCbCreate

Example This example demonstrates how to setup a callback object used in TipcSrvConn ErrorCbCreate Method:

In a separate class module, for example, GenericErrorCb

Implements TipcErrorCb

```
Private Sub TipcErrorCb_TipcError(ByVal c_err_num As Long, _
                                ByVal os_err_num As Long, _
                                ByVal socket_err_num As Long, _
                                ByVal smart_sockets_err_num As Long, _
                                ByVal arg As Variant)
```

```
Dim er As String, us As String
```

```
If (c_err_num <> 0) Then
    er = "C Error: " + Format(c_err_num)
```

```
End If
```

```
If (os_err_num <> 0) Then
    er = er + "OS Error: " + Format(os_err_num)
```

```
End If
```

```
If (socket_err_num <> 0) Then
    er = er + "Socket Error: " + Format(socket_err_num)
```

```
End If
```

```
If (smart_sockets_err_num <> 0) Then
    er = er + "Smart Socket Error: " +
        Format(smart_sockets_err_num)
```

```
End If
```

```
MsgBox "SmartSockets error!" + Chr(10) + er,
    vbDefaultButton1, "SS Error!"
```

```
End Sub
```

Now, This callback object would be used as follows:

```
Dim SrvConn as TipcSrvConn
```

```
Dim Svc as New TipcSvc
```

```
Dim Error_cb as New GenericErrorCb
```

```
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
```

```
SrvConn.Open
```

```
SrvConn.ErrorCbCreate(Error_cb, 2, Nothing)
```

```
SrvConn.Close
```

TipcEvent Class

TipcEvent — a container class for handling dispatcher events.

Description	The TipcEvent class provides a container class that handles events from the TipcDispatcher class. Objects of this TipcEvent class are instantiated from TipcDispatcher objects, and are used to remove events once created. When TipcEvents are created, they are attached to a dispatcher and called accordingly.
Methods	• Create Method • CreateMsg Method • CreateMsgType Method • CreateSocket Method • CreateTimer Method • Destroy Method
Properties	• CheckMode Property • Data Property • Dispatcher Property • Interval Property • Socket Property • Type Property
Events	None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcEvent Methods

Create Method

The Create method creates a user event to be used with a dispatcher.

Syntax *object.Create dispatcher, data, cb, event_arg*
where:
object is an object expression that evaluates to a TipcEvent object.
dispatcher is an object expression that evaluates to a TipcDispatcher object.
data is of type VARIANT to be passed to the dispatcher event.
cb is a user-defined callback object, implemented from the TipcEventCb class.
event_arg is a VARIANT user argument that is passed to the DispatcherEvent event in the user-defined callback class implemented from TipcEventCb.

Return Type None

C API TipcEventCreate

Example This example creates a user event:

```
' MyEventCb is a user class implemented from TipcEventCb
Dim event_cb as MyEventCb
Dim data as VARIANT

Dim t_event as new TipcEvent
Dim dispatcher as TipcDispatcher
Dim svc as new TipcSvc

Set dispatcher = svc.TipcDispatcherCreate
t_event.Create dispatcher, data, event_cb, Nothing
```

CreateMsg Method

The CreateMsg method creates a message event to be used with a dispatcher. This is invoked when a message is received on a connection with a particular subject.

Syntax *object.CreateMsg dispatcher, srv, subject, cb, event_arg*

where:

object is an object expression that evaluates to a TipcEvent object.

dispatcher is an object expression that evaluates to a TipcDispatcher object.

srv is an object expression that evaluates to a TipcSrvConn object.

subject is a string which is the subject that triggers the event.

cb is a user-defined callback object, implemented from the TipcEventMsgCb class.

event_arg is a VARIANT user argument that is passed to the DispatcherEvent event in the user-defined callback class implemented from TipcEventCb.

Return Type None

C API TipcEventCreateMsg

Example This example creates a message event and listens for messages:

```
' MyEventCb is a user class implemented from TipcEventMsgCb
Dim event_cb as MyEventCb
Dim data as VARIANT

Dim t_event as new TipcEvent
Dim dispatcher as TipcDispatcher
Dim svc as new TipcSvc
Dim srv as TipcSrvConn

Set dispatcher = svc.TipcDispatcherCreate
Set srv = svc.TipcSrvConnCreateNamed("Conn1")
srv.SubjectSetSubscribe "/SUBJECT", True
srv.Dispatcher = dispatcher
t_event.CreateMsg dispatcher, srv, "/SUBJECT", event_cb, Nothing
dispatcher.Mainloop 10
```

CreateMsgType Method

The CreateMsgType method creates a message type event to be used with a dispatcher.

Syntax *object.CreateMsgType dispatcher, srv, mt, cb, event_arg*
where:
object is an object expression that evaluates to a TipcEvent object.
dispatcher is an object expression that evaluates to a TipcDispatcher object.
srv is an object expression that evaluates to a TipcSrvConn object.
mt is an object expression that evaluates to a TipcMt object.
cb is a user-defined callback object, implemented from the
 TipcEventMsgCb class.
event_arg is a VARIANT user argument that is passed to the DispatcherEvent
 event in the user-defined callback class implemented from
 TipcEventCb.

Return Type None

C API TipcEventCreateMsgType

Example This example creates a message event and listens for messages:

```
' MyEventCb is a user class implemented from TipcEventMsgCb
Dim event_cb as MyEventCb
Dim data as VARIANT

Dim t_event as new TipcEvent
Dim dispatcher as TipcDispatcher
Dim svc as new TipcSvc
Dim srv as TipcSrvConn
Dim mt as TipcMt

Set dispatcher = svc.TipcDispatcherCreate
Set conn = svc.TipcSrvConnCreateNamed("Conn1")
Set Mt = svc.MtLookup("info")
srv.Dispatcher = dispatcher
t_event.CreateMsgType dispatcher, srv, mt, event_cb, Nothing
dispatcher.Mainloop 10
```


CreateSocket Method

The CreateSocket method creates a socket read or socket write event to be used with a dispatcher.

Syntax *object.Create dispatcher, socket, check_mode, cb, event_arg*

where:

object is an object expression that evaluates to a TipcEvent object.

dispatcher is an object expression that evaluates to a TipcDispatcher object.

socket is of type Long, which is a handle to a socket.

check_mode is of type TioCheckMode, either tioCheckRead, or tioCheckWrite.

cb is a user-defined callback object, implemented from the TipcEventCb class.

event_arg is a VARIANT user argument that is passed to the DispatcherEvent event in the user-defined callback class implemented from TipcEventCb.

Return Type None

C API TipcEventCreateSocket

Example This example creates a user event:

```
' MyEventCb is a user class implemented from TipcEventCb
Dim event_cb as MyEventCb
Dim socket as Long

Dim t_event as new TipcEvent
Dim dispatcher as TipcDispatcher
Dim svc as new TipcSvc

' get the socket

Set dispatcher = svc.TipcDispatcherCreate
t_event.CreateSocket dispatcher, socket, tioCheckRead, event_cb,
Nothing
```

CreateTimer Method

The CreateTimer method creates a timer event to be used with a dispatcher.

Syntax *object.CreateTimer dispatcher, interval, cb, event_arg*
where:
object is an object expression that evaluates to a TipcEvent object.
dispatcher is an object expression that evaluates to a TipcDispatcher object.
interval is of type Double to be passed to the dispatcher event, representing the interval that the callback will be invoked when the MainLoop method of the dispatcher is called.
cb is a user-defined callback object, implemented from the TipcEventCb class.
event_arg is a VARIANT user argument that is passed to the DispatcherEvent event in the user-defined callback class implemented from TipcEventCb.

Return Type None

C API TipcEventCreateTimer

Example This example creates a timer event:

```
' MyEventCb is a user class implemented from TipcEventCb
Dim event_cb as MyEventCb

Dim t_event as new TipcEvent
Dim dispatcher as TipcDispatcher
Dim svc as new TipcSvc

Set dispatcher = svc.TipcDispatcherCreate
' set the timer to be invoked every 3 seconds.
t_event.CreateTimer dispatcher, 3, event_cb, Nothing
```

Destroy Method

The Destroy method destroys an event.

Syntax *object.Destroy*
where *object* is an object expression that evaluates to a TipcEvent object.

Return Type None

C API TipcEventDestroy

Example This example destroys an event:

```
Dim t_event as new TipcEvent  
t_event.Destroy
```

TipcEvent Properties

CheckMode Property

The CheckMode property returns the check mode of a socket event. This is a read-only property.

Syntax *object*.CheckMode
where *object* is an object expression that evaluates to a TipcEvent object.

Data Type TioCheckMode

C API TipcEventGetCheckMode

Example This example returns the check mode of an event:

```
Dim t_event as new TipcEvent
...
if (t_event.CheckMode = tioCheckWrite) then
    debug.Print "CheckMode is write"
end if
```

Data Property

The Data property returns the user data for data events. This is a read-only property.

Syntax *object*.Data
where *object* is an object expression that evaluates to a TipcEvent object.

Data Type VARIANT

C API TipcEventGetData

Example This example gets the data from a user event:

```
Dim data as Variant
Dim t_event as New TipcEvent
...
Variant = t_event.Data
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Dispatcher Property

The Dispatcher property returns the dispatcher of an event.

Syntax `object.Dispatcher`

where *object* is an object expression that evaluates to a TipcEvent object.

Data Type TipcDispatcher

C API TipcEventGetDispatcher

Example This example gets the dispatcher of an event:

```
Dim dispatcher as TipcDispatcher
Dim t_event as New TipcEvent
...
dispatcher = t_event.Dispatcher
```

Interval Property

The Interval property sets or returns the interval of a TimerEvent.

Syntax `object.Interval [=interval_value]`

where:

object is an object expression that evaluates to a TipcEvent object.

interval_value is a Double which resets the interval.

Data Type Double

C API TipcEventGetInterval, TipcEventSetInterval

Example This example prints the interval of a timer event, then sets the interval to 5 seconds:

```
Dim interval as Double
Dim t_event as New TipcEvent
...
interval = t_event.Interval
Debug.Print "Interval is " + Str(interval)
t_event.Interval = 5.0
```

Socket Property

The Socket property returns the socket handle of a SocketEvent.

Syntax *object*.Socket
where *object* is an object expression that evaluates to a TipcEvent object.

Data Type Long (Integer)

C API TipcEventGetSocket

Example This example gets the socket handle of a socket event:

```
Dim socket as Long
Dim t_event as New TipcEvent
...
socket = t_event.Socket
```

Type Property

The Type property returns the type of an event.

Syntax *object*.Type
where *object* is an object expression that evaluates to a TipcEvent object.

Data Type TipcEventType

C API TipcEventGetType

Example This example gets the type of an event:

```
Dim t_event as new TipcEvent
...
if t_event.Type = TipcEventTimer then
    debug.Print "event type is Timer"
end if
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcEventCb Class

TipcEventCb — class used to derive a user class for dispatcher events.

Description	The TipcEventCb class provides an interface to derive user callbacks that are invoked through a dispatcher. The DispatcherEvent function must be overridden by the RTclient.
Methods	None
Properties	None
Events	• DispatcherEvent Event

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcEventCb Events

DispatcherEvent Event

The DispatcherEvent event is triggered based upon the type of event. User events (TipcEvent Create Method) are fired immediately, Socket events are fired upon a read (tioCheckRead), or write (tioCheckWrite) on a socket, and timer events are fired upon a specified interval.

Syntax

```
Private Sub TipcEventCb_DispatcherEvent _  
    (ByVal event_type As TIPCCOMLib.TipcEventType, _  
     ByVal data As Variant, _  
     ByVal Socket As Long, _  
     ByVal check_mode As TIPCCOMLib.TioCheckMode, _  
     ByVal Interval As Double, _  
     ByVal arg As Variant)
```

where:

event_type is the type of event being fired.

data is user data if this is being fired from a user event.

Socket is a socket handle if this is being fired from a socket event.

check_mode is the socket check mode if this is being fired from a socket event.

Interval is the interval of the event if this is a timer event.

arg is a user-defined variant, which is assigned through the TipcEvent Create Method, CreateSocket Method, or CreateTimer Method.

Return Type None

C API No comparable C API function

Example This example demonstrates how to setup an event callback object that can be passed to TipcEvent Create Method, CreateSocket Method, or CreateTimer Method:

In a separate class module, for example, GenericEventCb

Implements TipcEventCb


```

Private Sub TipcEventCb_DispatcherEvent _
    (ByVal event_type As TIPCCOMLib.TipcEventType, _
    ByVal data As Variant, _
    ByVal Socket As Long, _
    ByVal check_mode As TIPCCOMLib.TioCheckMode, _
    ByVal Interval As Double, _
    ByVal arg As Variant)

    Debug.Print "Inside EventCb"

End Sub

```

Now, This callback object would be used as follows:

```

Dim Svc as New TipcSvc
Dim event_cb as New GenericEventCb
Dim dispatcher as TipcDispatcher

Set dispatcher = svc.TipcDispatcherCreate()

' create a timer event to be fired every 2 seconds
Event.CreateTimer dispatcher, event_cb, 2, Nothing

Dispatcher.MainLoop 10

```

TipcEventMsgCb Class

TipcEventMsgCb — class for events in the TipcDispatcher Class that handle messages.

Description	The TipcEventMsgCb class provides an interface for handling events in the TipcDispatcher Class that handle messages.
Methods	None
Properties	None
Events	• DispatcherMsgEvent Event

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcEventMsgCb Events

DispatcherMsgEvent Event

The DispatcherMsgEvent event is triggered when a message is processed by the dispatcher. In the case of a Message Event, the message must match the subject in the creation of the event (see the TipcEvent CreateMsg Method). In the case of a message type event, the message must be of the type passed when the event was created (see the TipcEvent CreateMsgType Method).

Syntax

```
Private Sub TipcEventMsgCb_DispatcherMsgEvent( _
    ByVal event_type As TIPCCOMLib.TipcEventType, _
    ByVal pMsg As TIPCCOMLib.ITipcMsg, _
    ByVal arg As Variant)
```

where:

event_type is the type of event being fired.

pMsg is the TipcMsg object that has been received.

arg is a use- defined variant, which is assigned through the TipcEvent CreateMsg Method, or the TipcEvent CreateMsgType Method.

Return Type None

C API No comparable C API function

Example This example demonstrates how to setup an event callback object that can be passed to the TipcEvent CreateMsg Method or the TipcEvent CreateMsgType Method:

In a separate class module, for example, GenericMsgEventCb

Implements TipcEventMsgCb

```
Private Sub TipcEventMsgCb_DispatcherMsgEvent( _
    ByVal event_type As TIPCCOMLib.TipcEventType, _
    ByVal pMsg As TIPCCOMLib.ITipcMsg, _
    ByVal arg As Variant)
```

```
Dim mt As TipcMt
```

```
Set mt = pMsg.Type
```

```
Debug.Print "MsgEventCb firing. Event type is " + Str(event_type)
```

```
Debug.Print "MsgEventCb: Mt = " + mt.Name + ", Dest = " + _
    pMsg.Dest + ", Sender = " + pMsg.Sender
```

```
End Sub
```

Now, This callback object would be used as follows:

```
Dim Svc as New TipcSvc
Dim event_cb as New GenericMsgEventCb
Dim dispatcher as TipcDispatcher
Dim conn as TipcSrvConn
Dim mt as TipcMt

Set dispatcher = svc.TipcDispatcherCreate()
Set Conn = svc.TipcSrvConnCreate("", "", "", "")
Conn.Open
Conn.SubjectSetSubscribe "/foo", True
Conn.Flush

' Add the dispatcher to the conn
Conn.Dispatcher = dispatcher

' create a msg event to be fired when a conn receives
' a message on subject /Foo
Event.CreateMsg dispatcher, conn, "/Foo", event_cb, Nothing

' create a msg event to be fired anytime an info message
' is received. Note: If an info message is received on the
' /Foo subject, event_cb will be called twice
Set mt = svc.MtLookup("info")
Event.CreateMsg dispatcher, conn, mt, event_cb, Nothing

Dispatcher.MainLoop 10
```

TipcMsg Class

TipcMsg — class for constructing, manipulating and destroying SmartSockets messages

Description The TipcMsg class is the largest interface and is used to create, build up, decompose and set options for SmartSockets messages. See the *TIBCO SmartSockets User's Guide* for more information on message composition in SmartSockets RTclient applications.

Methods

- Ack Method
- AddNamedChar Method
- AddNamedInt2Array Method
- AddNamedInt4Array Method
- AddNamedInt8Array Method
- AddNamedMsgArray Method
- AddNamedReal4 Method
- AddNamedReal8 Method
- AddNamedReal16 Method
- AddNamedStr Method
- AddNamedTimestamp Method
- AddNamedXml Method
- AppendChar Method
- AppendInt2Array Method
- AppendInt4Array Method
- AppendInt8Array Method
- AppendMsgArray Method
- AppendReal4 Method
- AppendReal8 Method
- AppendReal16 Method
- AppendStr Method
- AppendTimestamp Method
- AppendUnknown Method
- Clone Method
- AddNamedBinary Method
- AddNamedInt2 Method
- AddNamedInt4 Method
- AddNamedInt8 Method
- AddNamedMsg Method
- AddNamedPersistedObject Method
- AddNamedReal4Array Method
- AddNamedReal8Array Method
- AddNamedReal16Array Method
- AddNamedStrArray Method
- AddNamedTimestampArray Method
- AppendBinary Method
- AppendInt2 Method
- AppendInt4 Method
- AppendInt8 Method
- AppendMsg Method
- AppendPersistedObject Method
- AppendReal4Array Method
- AppendReal8Array Method
- AppendReal16Array Method
- AppendStrArray Method
- AppendTimestampArray Method
- AppendXML Method
- CurrentType Method

- CurrentUnknown Method
- GetNamedBinary Method
- GetNamedInt2 Method
- GetNamedInt4 Method
- GetNamedInt8 Method
- GetNamedMsg Method
- GetNamedPersistedObject Method
- GetNamedReal4Array Method
- GetNamedReal8Array Method
- GetNamedReal16Array Method
- GetNamedStrArray Method
- GetNamedTimestampArray Method
- Load Method
- NextChar Method
- NextInt2Array Method
- NextInt4Array Method
- NextInt8Array Method
- NextMsgArray Method
- NextReal4 Method
- NextReal8 Method
- NextReal16 Method
- NextStr Method
- NextTimestamp Method
- NextType Method
- NextXml Method
- Save Method
- UpdateNamedBinary Method
- UpdateNamedInt2 Method
- UpdateNamedInt4 Method
- UpdateNamedInt8 Method
- UpdateNamedMsg Method
- UpdateNamedPersistedObject Method
- GetCompression Method
- GetNamedChar Method
- GetNamedInt2Array Method
- GetNamedInt4Array Method
- GetNamedInt8Array Method
- GetNamedMsgArray Method
- GetNamedReal4 Method
- GetNamedReal8 Method
- GetNamedReal16 Method
- GetNamedStr Method
- GetNamedTimestamp Method
- GetNamedXml Method
- NextBinary Method
- NextInt2 Method
- NextInt4 Method
- NextInt8 Method
- NextMsg Method
- NextPersistedObject Method
- NextReal4Array Method
- NextReal8Array Method
- NextReal16Array Method
- NextStrArray Method
- NextTimestampArray Method
- NextUnknown Method
- Reset Method
- SetCompression Method
- UpdateNamedCharacter Method
- UpdateNamedInt2Array Method
- UpdateNamedInt4Array Method
- UpdateNamedInt8Array Method
- UpdateNamedMsgArray Method
- UpdateNamedReal4 Method

- UpdateNamedReal4Array Method
- UpdateNamedReal8Array Method
- UpdateNamedReal16Array Method
- UpdateNamedStrArray Method
- UpdateNamedTimestampArray Method
- UpdateNamedReal8 Method
- UpdateNamedReal16 Method
- UpdateNamedStr Method
- UpdateNamedTimestamp Method
- UpdateNamedXML Method

Properties

- ArrivalTimestamp Property
- Current Property
- DeliveryMode Property
- Dest Property
- HeaderStrEncode Property
- NumFields Property
- ReplyTo Property
- SeqNum Property
- UserProp Property
- CorrelationID Property
- CurrentUnknown Property
- DeliveryTimeout Property
- Fields Property
- LbMode Property
- Priority Property
- Sender Property
- Type Property

Events None

TipcMsg Class Methods

Ack Method

The Ack method acknowledges the delivery of a message.

Syntax *object*.Ack
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type None

C API TipcMsgAck

Example This example acknowledges a message:

```
Dim msg s TipcMsg
set msg = TipcSvc1.Srv.MsgNext(2.0)
if NOT msg is Nothing Then
    msg.ack
End If
```

AddNamedBinary Method

The AddNamedBinary method appends a binary field to a message using a name.

Syntax *object*.AddNamedBinary *name*, *binary_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
binary_data is an array of bytes containing the binary data to add to a message.

Return Type None

C API TipcMsgAddNamedBinary

Example This example adds a named binary field to a message:

```
Dim blob(100) As byte
Dim i As Long
For i =1 to 64
```



```

        Blob(i) = Chr(i Mod 255)
    Next i
    msg.AddNamedBinary "binary_data", blob

```

AddNamedChar Method

The AddNamedChar method adds a named character field (field type CHAR, which is a 1-byte integer) to a message. The actual data type passed is a byte.

Syntax *object*.AddNamedChar *name*, *char_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

char_data is the data to add to a message.

Return Type None

C API TipcMsgAddNamedChar

Example This example appends a character field to a message:

```

Dim ch As byte
ch = Asc("C")
msg.AddNamedChar "char_data" ch

```

AddNamedInt2 Method

The AddNamedInt2 method adds a named 2-byte integer field (field type INT2) to a message.

Syntax *object*.AddNamedInt2 *name*, *int2_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

int2_data is the 2-byte integer data to add to a message.

Return Type None

C API TipcMsgAddNamedInt2

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example appends a named 2-byte integer field to a message:

```
Dim i2 As Integer
i2 =33
msg.AddNamedInt2 "int2_data", i2
```

AddNamedInt2Array Method

The AddNamedInt2Array method adds a named 2-byte integer array field (field type INT2_ARRAY) to a message.

Syntax *object*.AddNamedInt2Array *name*, *int2_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
int2_array_data is an integer array that specifies the array of 2-byte integers to add to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAddNamedInt2Array

Example This example appends a named 2-byte integer array field to a message:

```
Dim i2_array(16) As Integer
Dim i As Long
for i =0 to 15
i2_array(i) = i
Next i
msg.AddNamedInt2Array "int2Array_data", i2_array
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

AddNamedInt4 Method

The AddNamedInt4 method adds a named 4-byte integer field (field type INT4) to a message.

Syntax *object*.AddNamedInt4 *name*, *int4_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

int4_data is a long integer that specifies the 4-byte integer data to add to a message.

Return Type None

C API TipcMsgAddNamedInt4

Example This example appends a named 4-byte integer field to a message:

```
Dim i4 As Long
i4 = 123456789
msg.AddNamedInt4 "Int4_data", i4
```

AddNamedInt4Array Method

The AddNamedInt4Array method adds a named 4-byte integer array field (field type INT4_ARRAY) to a message.

Syntax *object*.AddNamedInt4Array *name*, *int4_array_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

int4_array_data is a long array that specifies the array of 4-byte integers to add to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAddNamedInt4Array

Example This example appends a named 4-byte integer array field to a message:

```
Dim i4_array(16) As Long
Dim i As Long
For i =0 to 15
i4_array(i) = i
N i
msg.AddNamedInt4Array "Int4_Array_data", i4_array
```

AddNamedInt8 Method

The AddNamedInt8 method adds a named 8-byte integer field (field type INT8) to a message.

Syntax *object.AddNamedInt8 name, int8_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
int8_data is a type of Currency (CY) that specifies the 8-byte integer data to add to a message.

Return Type None

C API TipcMsgAddNamedInt8

Example This example appends a named 8-byte integer field to a message:

```
Dim i8 As Currency
i8 = 123456789
msg.AddNamedInt8 "Int8_data", i8
```

AddNamedInt8Array Method

The AddNamedInt8Array method adds a named 8-byte integer array field (field type INT8_ARRAY) to a message.

Syntax *object.AddNamedInt8Array name, int8_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.

int8_array_data is a long array that specifies the array of 8-byte integers to add to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAddNamedInt8Array

Example This example appends a named 8-byte integer array field to a message:

```
Dim i8_array(16) As Currency
Dim i As Long
For i = 0 to 15
    i8_array(i) = i
Next i
msg.AddNamedInt8Array "Int8_Array_data", i8_array
```

AddNamedMsg Method

The AddNamedMsg method adds a named message field (field type MSG) to a message.

Syntax *object.AddNamedMsg name, msg_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

msg_data is a TipcMsg class message string to add to the object message.

Return Type None

C API TipcMsgAddNamedMsg

Example This example appends a named message field to a message:

```
Dim msg2 As TipcMsg
Dim mt As TipcMt
Set mt = TipcSvc1.MtLookup("control")
Set msg2 = TipcSvc1.MsgCreate(mt)
msg2.AddNamedStr "quit force"
msg.AddNamedMsg "msg_data", msg2
```

AddNamedMsgArray Method

The AddNamedMsgArray method adds a named message array field (field type MSG_ARRAY) to a message.

Syntax *object*.AddNamedMsgArray *name*, *msg_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
msg_array_data is a TipcMsgArray class that specifies the TipcMsg objects to add to the object message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAddNamedMsgArray

Example This example appends a named message array field to a message:

```
Dim msg_array(4) As TipcMsg
Dim i As Long
Dim mt As TipcMt
Set mt = TipcSvc1.MtLookup("info")
For i =0 to 3
Set msg_array(i) = TipcSvc1.MsgCreate(mt)
msg_array(i).AppendStr "Msg Number: " & Format(i)
Next i
msg.AddNamedMsgArray "msg_array_data", msg_array
```

AddNamedPersistedObject Method

The AddNamedPersistedObject method adds a user-defined object as a blob to a message. Internally this is stored as binary data in the message, but uses the IPersistStream interface to save the message to a blob first.

Syntax *object*.AddNamedPersistedObject *name*, *user_object*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
user_object is a user object that implemented the IPersistStream interface to save the object.

Return Type None

C API No comparable C API function

Example This example retrieves the value of the current persisted object field in the message:

```
Set contents = svc.MsgCreate(svc.MtLookup("info"))
contents.AppendStr "my string"
contents.Dest = "/mydest"
msg.AddNamedPersistedObject "fred", contents
```

AddNamedReal4 Method

The AddNamedReal4 method adds a named 4-byte real number field (field type REAL4) to a message.

Syntax *object.AddNamedReal4 name, real4_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

real4_data is a single real number that specifies the 4-byte real number to add to a message.

Return Type None

C API TipcMsgAddNamedReal4

Example This example appends a named 4-byte real field to a message:

```
Dim r4 As Single
r4 = 123.45
msg.AddNamedReal4 "real4_data", r4
```

AddNamedReal4Array Method

The AddNamedReal4Array method adds a named 4-byte real number array field (field type REAL4_ARRAY) to a message.

Syntax *object.AddNamedReal4Array name, real4_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
real4_array_data is a single array that specifies the array of 4-byte real numbers to add to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAddNamedReal4Array

Example This example appends a 4-byte real array field to a message:

```
Dim r4_array(16) As Single
Dim i As Long
For i =0 to 15
r4_array(i) = 1234.123 + i
Next i
msg.AddNamedReal4Array "real4_array_data", r4_array
```

AddNamedReal8 Method

The AddNamedReal8 method adds a named 8-byte real number field (field type REAL8) to a message.

Syntax *object.AddNamedReal8 name, real8_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
real8_data is the 8-byte real number to add to a message.

Return Type None

C API TipcMsgAddNamedReal8

Example This example adds a named 8-byte real field to a message:

```
Dim r8 As Double
r8 = 1234567.123456
msg.AddNamedReal8 "real8_data", r8
```

AddNamedReal8Array Method

The AddNamedReal8Array method adds a named 8-byte real number array field (field type REAL8_ARRAY) to a message.

Syntax *object.AddNamedReal8Array name, real8_array_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

real8_array_data is a double array that specifies the array of 8-byte real numbers to add to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAddNamedReal8Array

Example This example appends an 8-byte real field to a message:

```
Dim r8_array(16) As Double
Dim i As Long
For i = 0 to 15
    r8_array(i) = 1234567.123456 + i
Next i
msg.AddNamedReal8Array "real8_array_data", r8_array
```

AddNamedReal16 Method

The AddNamedReal16 method adds a named 16-byte real number field (field type REAL16) to a message.

Syntax *object.AddNamedReal16 name, real16_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to add to a message.

real16_data is the 16-byte real number to add to a message. This is implemented as an array of 16 bytes.

Return Type None

C API TipcMsgAddNamedReal16

Example This example adds a named 16-byte real field to a message:

```
Dim r16(16)As byte
' Fill in array
msg.AddNamedReal16 "real16_data", r16
```

AddNamedReal16Array Method

The AddNamedReal16Array method adds a named 16-byte real number array field (field type REAL16_ARRAY) to a message.

Syntax *object.AddNamedReal16Array name, real16_array_data*
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
real16_array_data is a double array that specifies the array of 16-byte real numbers to add to a message. This is passed as a SAFEARRAY, containing arrays of 16 bytes.

Return Type None

C API TipcMsgAddNamedReal16Array

Example This example appends a 16-byte real field to a message:

```
Dim r16_array(10)(16) As Byte
' fill in array here
msg.AddNamedReal16Array "real16_array_data", r16_array
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

AddNamedStr Method

The AddNamedStr method adds a named null-terminated character string field (field type STR) to a message. A string field is a special case of an array of characters.

Syntax *object.AddNamedStr name, str_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
str_data is the string data to add to a message.

Return Type None

C API TipcMsgAddNamedStr

Example This example appends a named string field to a message:

```
Dim s As String  
s = "Test String")  
msg.AddNamedStr "string_data", s
```

AddNamedStrArray Method

The AddNamedStrArray method adds a named string array field (field type STR_ARRAY) to the message.

Syntax *object.AddNamedStrArray name, str_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
str_array_data is a string array that specifies the array of string data to be added to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAddNamedStrArray

Example This example appends a named string array field to a message:

```
Dim s_array(8) As String
Dim i As Long
For i =0 to 7
s_array(i) = "Test String" & Format(i)
Next i
msg.AddNamedStrArray "string_array_data", s_array
```

AddNamedTimestamp Method

The AddNamedTimestamp method adds a named timestamp field (field type `TIMESTAMP`) to a message. The timestamp is the current computer wall clock time in the number of seconds since midnight January 1, 1970. The `TIMESTAMP` field is an eight-byte real number containing *timestamp_data* to the end of a message's data.

Syntax *object*.AddNamedTimestamp *name*, *timestamp_data*
where:
object is an object expression that evaluates to a `TipcMsg` object.
name is a string that specifies the field name to add to a message.
timestamp_data is the 8-byte real number to add to a message.

Return Type None

C API `TipcMsgAddNamedTimestamp`

Example This example adds a named timestamp field to a message:

```
Dim timestamp As Double
timestamp = 1234567.123456
msg.AddNamedTimestamp "timestamp_data", timestamp
```

AddNamedTimestampArray Method

The AddNamedTimestampArray method adds a named timestamp array field (field type `TIMESTAMP_ARRAY`) to a message. The timestamp is the current computer wall clock time in the number of seconds since midnight January 1, 1970. The `TIMESTAMP_ARRAY` field is an eight-byte real number array containing *timestamp_array_data* to the end of a message's data.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Syntax `object.AddNamedTimestampArray name, timestamp_array_data`
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
timestamp_array_data is a double array that specifies the array of timestamps to add to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAddNamedTimestampArray

Example This example appends a named timestamp array field to a message:

```
Dim timestamp_array(16) As Double
Dim i As Long
For i = 0 to 15
    timestamp_array(i) = 1234567.123456 + i
Next i
msg.AddNamedTimestampArray "timestamp_array_data", timestamp_array
```

AddNamedXml Method

The AddNamedXml method adds a named TipcXml object (field type TIPCXML) to a message.

Syntax `object.AddNamedXml name, xml_object`
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add to a message.
xml_object is the TipcXml object to add to a message.

Return Type None

C API TipcMsgAddNamedXml

Example This example appends a named 4-byte integer field to a message:

```
Dim tXml As TipcXml
Dim tSvc as New TipcSvc
Set tXml = tSvc.XmlCreate("<data><data1><temp=hot></data1></data>")
msg.AddNamedXml "Xml_data", tXml
```

AppendBinary Method

The AppendBinary method appends a binary field to a message. A binary field has the field type BINARY, which is a non-restrictive array of characters. Binary fields are useful for transferring images or large amounts of opaque data.

Syntax *object.AppendBinary binary_data*
where:
object is an object expression that evaluates to a TipcMsg object.
binary_data is an array of bytes containing the binary data to add to a message.

Return Type None

C API TipcMsgAppendBinary

Example This example appends a binary field to a message:

```
Dim blob (64) As byte
Dim i As Long

For i = 0 to 63
    Blob(i) = Chr(i Mod 255)
Next i
msg.AppendBinary blob
```

AppendChar Method

The AppendChar method appends a character field (field type CHAR, which is a 1-byte integer) to a message. The actual COM data type passed is a 2-byte integer to better match the common Chr() and Asc() functions' data type.

Syntax *object.AppendChar char_data*
where:
object is an object expression that evaluates to a TipcMsg object.
char_data is a integer that specifies the 1-byte of data to append to a message.

Return Type None

C API TipcMsgAppendChar

Example This example appends a character field to a message:

```
Dim ch As Integer
ch = Asc("C")
msg.AppendChar ch
```

AppendInt2 Method

The AppendInt2 method appends a 2-byte integer field (field type INT2) to a message.

Syntax *object.AppendInt2 int2_data*

where:

object is an object expression that evaluates to a TipcMsg object.

int2_data is a integer that specifies the 2-byte integer data to append to a message.

Return Type None

C API TipcMsgAppendInt2

Example This example appends a 2-byte integer field to a message:

```
Dim i2 As Integer
i2 = 33
msg.AppendInt2 i2
```

AppendInt2Array Method

The AppendInt2Array method appends a 2-byte integer array field (field type INT2_ARRAY) to a message.

Syntax *object.AppendInt2Array int2_array_data*

where:

object is an object expression that evaluates to a TipcMsg object.

int2_array_data is an integer array that specifies the array of 2-byte integers to append to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAppendInt2Array

Example This example appends a 2-byte integer array field to a message:

```
Dim i2_array(16) As Integer
Dim i As Long

for i = 0 to 15
    i2_array(i) = i
Next i
msg.AppendInt2Array i2_array
```

AppendInt4 Method

The AppendInt4 method appends a 4-byte integer field (field type INT4) to a message.

Syntax *object.AppendInt4 int4_data*
where:
object is an object expression that evaluates to a TipcMsg object.
int4_data is a long integer that specifies the 4-byte integer data to append to a message.

Return Type None

C API TipcMsgAppendInt4

Example This example appends a 4-byte integer field to a message:

```
Dim i4 As Long

i4 = 123456789
msg.AppendInt4 i4
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

AppendInt4Array Method

The AppendInt4Array method appends a 4-byte integer array field (field type INT4_ARRAY) to a message.

Syntax *object.AppendInt4Array int4_array_data*

where:

object is an object expression that evaluates to a TipcMsg object.

int4_array_data is an integer array that specifies the array of 4-byte integers to append to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAppendInt4Array

Example This example appends a 4-byte integer array field to a message:

```
Dim i4_array(16) As Long
Dim i As Long
For i = 0 to 15
    i4_array(i) = i
Next i
msg.AppendInt4Array i4_array
```

AppendInt8 Method

The AppendInt8 method appends an 8-byte integer field (field type INT8) to a message.

Syntax *object.AppendInt8 int8_data*

where:

object is an object expression that evaluates to a TipcMsg object.

int8_data is a type of Currency (CY) that specifies the 8-byte integer data to add to a message.

Return Type None

C API TipcMsgAppendInt8

Example This example appends an 8-byte integer field to a message:

```
Dim i8 As Currency
i8 = 123456789
msg.AppendInt8 i8
```

AppendInt8Array Method

The AppendInt8Array method appends an 8-byte integer array field (field type INT8_ARRAY) to a message.

Syntax *object.AppendInt8Array int8_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
int8_array_data is a Currency (CY) array that specifies the array of 8-byte integers to append to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAppendInt8Array

Example This example appends an 8-byte integer array field to a message:

```
Dim i8_array(16) As Currency
Dim i As Long
For i = 0 to 15
    i8_array(i) = i
Next i
msg.AppendInt8Array i8_array
```

AppendMsg Method

The AppendMsg method appends a message field (field type MSG) to a message.

Syntax *object.AppendMsg msg_data*
where:
object is an object expression that evaluates to a TipcMsg object.
msg_data is a TipcMsg class message string to append to the object message.

Return Type None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

C API TipcMsgAppendMsg

Example This example appends a message field to a message:

```
Dim msg2 As TipcMsg
Dim mt As TipcMt

Set mt = TipcSvc1.MtLookup("control")
Set msg2 = TipcSvc1.MsgCreate(mt)
msg2.AppendStr "quit force"
msg.AppendMsg msg2
```

AppendMsgArray Method

The AppendMsgArray method appends a message array field (field type MSG_ARRAY) to a message.

Syntax *object*.AppendMsgArray *msg_array_data*

where:

object is an object expression that evaluates to a TipcMsg object.

msg_array_data is a TipcMsgArray class with the TipcMsg objects to append to the object message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAppendMsgArray

Example This example appends a message array field to a message:

```
Dim msg_array(4) As TipcMsg
Dim i As Long
Dim mt As TipcMt

Set mt = TipcSvc1.MtLookup("info")
For i = 0 to 3
    Set msg_array(i) = TipcSvc1.MsgCreate(mt)
    msg_array(i).AppendStr "Msg Number: " & Format(i)
Next i
msg.AppendMsgArray msg_array
```

AppendPersistedObject Method

The AppendPersistedObject method appends a user-defined object as a blob to a message. Internally this is stored as binary data in the message, but uses the IPersistStream interface to save the message to a blob first.

Syntax *object.AppendPersistedObject user_object*
where:
object is an object expression that evaluates to a TipcMsg object.
user_object is a user object that has implemented the IPersistStream interface to save the object.

Return Type None

C API No comparable C API function

Example This example appends a persisted object to the message:

```
Set contents = svc.MsgCreate(svc.MtLookup("info"))
contents.AppendStr "my string"
contents.Dest = "/mydest"
msg.AppendPersistedObject contents
```

AppendReal4 Method

The AppendReal4 method appends a 4-byte real number field (field type REAL4) to a message.

Syntax *object.AppendReal4 real4_data*
where:
object is an object expression that evaluates to a TipcMsg object.
real4_data is a single real number that specifies the 4-byte real number to append to a message.

Return Type None

C API TipcMsgAppendReal4

Example This example appends a 4-byte real field to a message:

```
Dim r4 As Single
r4 = 123.45
msg.AppendReal4 r4
```

AppendReal4Array Method

The AppendReal4Array method appends a 4-byte real number array field (field type REAL4_ARRAY) to a message.

Syntax	<i>object.AppendReal4Array real4_array_data</i> where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>real4_array_data</i> is a single array that specifies the array of 4-byte real numbers to append to a message. This is passed as a SAFEARRAY.
Return Type	None
C API	TipcMsgAppendReal4Array
Example	This example appends a 4-byte real array field to a message: <pre>Dim r4_array(16) As Single Dim i As Long For i = 0 to 15 r4_array(i) = 1234.123 + i Next i msg.AppendReal4Array r4_array</pre>

AppendReal8 Method

The AppendReal8 method appends an 8-byte real number field (field type REAL8) to a message.

Syntax	<i>object.AppendReal8 real8_data</i> where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>real8_data</i> is a double real number that specifies the 8-byte real number to append to a message.
Return Type	None
C API	TipcMsgAppendReal8

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example appends an 8-byte real field to a message:

```
Dim r8 As Double
r8 = 1234567.123456
msg.AppendReal8 r8
```

AppendReal8Array Method

The AppendReal8Array method appends an 8-byte real number array field (field type REAL8_ARRAY) to a message.

Syntax *object.AppendReal8Array real8_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
real8_array_data is a double array that specifies the array of 8-byte real numbers to append to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAppendReal8Array

Example This example appends an 8-byte real field to a message:

```
Dim r8_array(16) As Double
Dim i As Long

For i = 0 to 15
    r8_array(i) = 1234567.123456 + i
Next i
msg.AppendReal8Array r8_array
```

AppendReal16 Method

The AppendReal16 method appends a 16-byte real number field (field type REAL16) to a message.

Syntax *object.AppendReal16 real16_data*
where:
object is an object expression that evaluates to a TipcMsg object.
real16_data is the 16-byte real number to add to a message. This is implemented as an array of 16 bytes.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Return Type	None
C API	TipcMsgAppendReal16
Example	This example appends a 16-byte real field to a message:<pre>Dim r16(16) As Byte ' Fill in array msg.AppendReal16 r16</pre>

AppendReal16Array Method

The AppendReal16Array method appends a 16-byte real number array field (field type REAL16_ARRAY) to a message.

Syntax *object*.AppendReal16Array *real16_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
real16_array_data is a double array that specifies the array of 16-byte real numbers to append to a message. This is passed as a SAFEARRAY. Each element of the array is an array of 16 bytes.

Return Type	None
C API	TipcMsgAppendReal16Array
Example	This example appends a 16-byte real field to a message:<pre>Dim r16_array(10)(16) As Byte Dim i As Long For i = 0 to 15 ' fill in r16_array(i) Next i msg.AppendReal16Array r16_array</pre>

AppendStr Method

The AppendStr method appends a null-terminated character string field (field type STR) to a message. A string field is a special case of an array of characters.

Syntax *object.AppendStr str_data*
 where:
object is an object expression that evaluates to a TipcMsg object.
str_data is the string data to append to a message.

Return Type None

C API TipcMsgAppendStr

Example This example appends a string field to a message:

```
Dim s As String
s = "Test String")
msg.AppendStr s
```

AppendStrArray Method

The AppendStrArray method appends a string array field (field type STR_ARRAY) to a message.

Syntax *object.AppendStrArray str_array_data*
 where:
object is an object expression that evaluates to a TipcMsg object.
str_array_data is a string array that specifies the array of string data to append to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAppendStrArray

Example This example appends a string array field to a message:

```
Dim s_array(8) As String
Dim i As Long

For i = 0 to 7
    s_array(i) = "Test String" & Format(i)
Next i
msg.AppendStrArray s_array
```

AppendTimestamp Method

The AppendTimestamp method appends a timestamp field (field type `TIMESTAMP`) to a message. The timestamp is the current computer wall clock time in the number of seconds since midnight January 1, 1970. The `TIMESTAMP` field is an eight-byte real number containing *timestamp_data* to the end of a message's data.

Syntax `object.AppendTimestamp timestamp_data`

where:

object is an object expression that evaluates to a `TipCMsg` object.

timestamp_data is a double real number that specifies the timestamp to append to a message.

Return Type None

C API `TipCMsgAppendTimestamp`

Example This example appends a timestamp field to a message:

```
Dim timestamp As Double
timestamp = 1234567.123456
msg.AppendTimestamp timestamp
```

AppendTimestampArray Method

The AppendTimestampArray method appends a timestamp array field (field type `TIMESTAMP_ARRAY`) to a message. The timestamp is the current computer wall clock time in the number of seconds since midnight January 1, 1970. The `TIMESTAMP_ARRAY` field is an eight-byte real number array containing *timestamp_array_data* to the end of a message's data.

Syntax *object.AppendTimestampArray timestamp_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
timestamp_array_data is a double array that specifies the array of timestamps to
append to a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgAppendTimestampArray

Example This example appends a timestamp array field to a message:

```
Dim timestamp_array(16) As Double
Dim i As Long
For i = 0 to 15
timestamp_array(i) = 1234567.123456 + i
Next i
msg.AppendTimestampArray timestamp_array
```

AppendUnknown Method

The AppendUnknown method appends a field with a value that is not known. In other words, the value is unset or n/a.

Syntax *object.AppendUnknown field_type*
where:
object is an object expression that evaluates to a TipcMsg object.
field_type is an integer that specifies the type of the new field.

Return Type None

Remarks Frequently, messages have a fixed number of fields in a constant order. In that case, omitting a field when data does not exist is not feasible. Setting the value of the field to unknown allows fields to be marked as having no data. This is especially valuable when the entire range of field values is valid within an application and there is need to denote that the field has no valid data.

C API TipcMsgAppendUnknown

Example This example appends a field of value unknown to a message:

```
msg.AppendUnknown 7 'int4 type
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

AppendXML Method

The AddNamedXml method appends a TipcXml object (field type TIPCXML) to a message.

Syntax `object.AppendXml xml_object`

where:

object is an object expression that evaluates to a TipcMsg object.

xml_object is the TipcXml object to add to a message.

Return Type None

C API TipcMsgAddNamedXml

Example This example appends a named 4-byte integer field to a message:

```
Dim Tsvc As new TipcSvc
Dim tXml As TipcXml
Set Txml = Tsvc.Create("<data><data1><temp=hot/></data1></data>")
txmlmsg.AddNamedXml tXml
```

Clone Method

The Clone method makes an identical copy of the TipcMsg message object.

Syntax `object.Clone`

where *object* is an object expression that evaluates to a TipcMsg object.

Return Type TipcMsg

C API TipcMsgClone

Example This example makes an identical copy of the message object:

```
Dim cmsg As New TipcMsg
Set cmsg = msg.Clone
```

CurrentType Method

The CurrentType method retrieves the current field's field type in a message.

Syntax *object*.CurrentType
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type TipcFt

C API TipcMsgGetCurrentFieldType

Example This example retrieves the field type of the current field in a message:

Dim ft As TipcFt
ft = msg.CurrentType

CurrentUnknown Method

The CurrentUnknown method returns True if the value of the current field of a message is unknown or unset.

Syntax *object*.CurrentUnknown
where *object* is an object expression that evaluates to a TipcMsg object.

Data Type Boolean

C API TipcMsgGetCurrentFieldUnknown

Example This example determines if the current field of the message is of unknown value:

Dim fieldUnk As Boolean
fieldUnk = msg.CurrentUnknown

GetCompression Method

The GetCompression method returns the compression property of the message.

Syntax *object*.GetCompression
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type Boolean

C API TipcMsgGetCompression

Example This example gets the compression property of a message:

```
Dim msgOut As TipcMsg
Dim compression As Boolean
Dim Tsvc As New TipcSvc

Set msgOut = Tsvc.MsgCreate(Tsvc.MtLookup("string_data"))
msgOut.Dest = "_all"

msgOut.SetCompression True
compression = msgOut.GetCompression
MsgBox(compression)
```

GetNamedBinary Method

The GetNamedBinary method retrieves the value of a binary field of a message using a name.

Syntax *object*.GetNamedBinary (*name*, *binary_data*, *data_size*)

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to retrieve from a message.

binary_data is an array of bytes that specifies the binary data to retrieve from a message.

data_size is the size of the returned array of bytes.

Return Type None

C API TipcMsgGetNamedBinary

Example This example retrieves the value of a named binary field in a message:

```
Dim blob As String
Dim iRetBytes As Integer
msg.GetNamedBinary("binary_data", blob, iRetBytes)
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

GetNamedChar Method

The GetNamedChar method retrieves the value of a character field in a message, using a name. If necessary, the field is converted to the data format used by the process.

Syntax *object*.GetNamedChar (*name*)
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.

Return Type Char

C API TipcMsgGetNamedChar

Example This example retrieves the value of a named character field in the message:

```
Dim ch As Integer
ch = msg.GetNamedChar("char_data")
```

GetNamedInt2 Method

The GetNamedInt2 method retrieves the value of a named field in a message, which must be a 2-byte integer field. If necessary, the field is converted to the data format used by the process.

Syntax *object*.GetNamedInt2 (*name*)
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.

Return Type Integer

C API TipcMsgGetNamedInt2

Example This example retrieves the value of a named 2-byte integer field of the message:

```
Dim i2 As Integer
i2 = msg.GetNamedInt2("int2_data")
```

GetNamedInt2Array Method

The GetNamedInt2Array method retrieves the value of a named field in a message, which must be an array of 2-byte integers. If necessary, the field is converted to the data format used by the process.

Syntax	<code>object.GetNamedInt2Array (name, int2_array_data, array_size)</code> where: <code>object</code> is an object expression that evaluates to a TpcMsg object. <code>name</code> is a string that specifies the field name to retrieve from a message. <code>int2_array_data</code> is an integer array used to hold the array of 2-byte integers to retrieve from a message. This is passed as a SAFEARRAY. <code>array_size</code> is a long integer that specifies the number of elements placed in <code>int2_array_data</code>.
Return Type	None
Remarks	<code>array_size</code> indicates how many actual items were placed into <code>int2_array_data</code> .
C API	TpcMsgGetNamedInt2Array
Example	This example retrieves the value of a named 2-byte integer array field of the message: <pre>Dim numElements As Long Dim i2_array(12) As Integer msg.GetNamedInt2Array("i2_array_data", i2_array, numElements)</pre>

GetNamedInt4 Method

The GetNamedInt4 method retrieves the value of a named field in a message, which must be a 4-byte integer field. If necessary, the field is converted to the data format used by the process.

Syntax	<code>object.GetNamedInt4 (name)</code> where: <code>object</code> is an object expression that evaluates to a TpcMsg object. <code>name</code> is a string that specifies the field name of the field to retrieve from a message.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Return Type	Long (Integer)
C API	TipcMsgGetNamedInt4
Example	This example retrieves the value of a named 4-byte integer field of the message: <pre>Dim i4 As Long i4 = msg.GetNamedInt4("int4_data")</pre>

GetNamedInt4Array Method

The GetNamedInt4Array retrieves the value of a named field in a message, which must be an array of 4-byte integers. If necessary, the field is converted to the data format used by the process.

Syntax	<i>object</i> .GetNamedInt4Array (<i>name</i> , <i>int4_array_data</i> , <i>array_size</i>) where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>name</i> is a string that specifies the field name to retrieve from a message. <i>int4_array_data</i> is a long integer array used to hold the data retrieved from the message. This is passed as a SAFEARRAY. <i>array_size</i> is a long integer that specifies the number of elements placed in <i>int4_array_data</i> .

Return Type	Long (Integer)
Remarks	<i>array_size</i> indicates how many actual items were placed into <i>int4_array_data</i> .
C API	TipcMsgGetNamedInt4Array
Example	This example retrieves the value of a named 4-byte integer array field of the message: <pre>Dim numElements As Long Dim i4_array(12) As Long msg.GetNamedInt4Array("int4_array_data", i4_array, numElements)</pre>

GetNamedInt8 Method

The GetNamedInt8 method retrieves the value of a named field in a message, which must be an 8-byte integer field. If necessary, the field is converted to the data format used by the process.

Syntax	<i>object</i>.GetNamedInt8 (<i>name</i>) where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>name</i> is a string that specifies the field name of the field to retrieve from a message.
Return Type	Currency (CY)
C API	TipcMsgGetNamedInt8
Example	This example retrieves the value of a named 8-byte integer field of the message: <pre>Dim i8 As Currency i8 = msg.GetNamedInt8("int8_data")</pre>

GetNamedInt8Array Method

The GetNamedInt8Array retrieves the value of a named field in a message, which must be an array of 8-byte integers. If necessary, the field is converted to the data format used by the process.

Syntax	<i>object</i>.GetNamedInt8Array (<i>name</i>, <i>int8_array_data</i>, <i>array_size</i>) where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>name</i> is a string that specifies the field name to retrieve from a message. <i>int8_array_data</i> is a long integer array used to hold the data retrieved from the message. This is passed as a SAFEARRAY containing Currency (CY) values. <i>array_size</i> is a long integer that specifies the number of elements placed in <i>int8_array_data</i>.
Return Type	None
Remarks	<i>array_size</i> value indicates how many actual items were placed into <i>int8_array_data</i> .

C API TipcMsgGetNamedInt8Array

Example This example retrieves the value of a named 8-byte integer array field of the message:

```
Dim numElements As Long
Dim i8_array(12) As Long
msg.GetNamedInt4Array("int8_array_data", i8_array, numElements)
```

GetNamedMsg Method

The GetNamedMsg method retrieves the value of a named field in a message, which must be a message field.

Syntax *object*.GetNamedMsg (*name*)
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.

Return Type TipcMsg

C API TipcMsgGetNamedMsg

Example This example retrieves the value of a named message field of the message:

```
Dim msg2 As TipcMsg
msg2 = msg.GetNamedMsg("msg_data")
```

GetNamedMsgArray Method

The GetNamedMsgArray method retrieves the value of a named field in a message, which must be an array of messages.

Syntax *object*.GetNamedMsgArray (*name*, *msg_array_data*, *array_size*)
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.
msg_array_data is class TipcMsgArray, and is used to hold the data retrieved from the message. This is passed as a SAFEARRAY.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

array_size is a long integer that specifies the number of elements placed in *msg_array_data*.

Return Type None

C API TipcMsgGetNamedMsgArray

Example This example retrieves the value of a named message-array field of the message:

```
Dim numElements As Long
Dim msg_array(8) As TipcMsg
Dim msgName As String
  Dim i As Long
  msg.GetNamedMsgArray("msg_array_data", msg_array, numElements)
  For i =0 to numElements - 1
    msgName = msg_array(i).Type.Name
  Next i
```

GetNamedPersistedObject Method

The GetNamedPersistedObject method returns a user-defined object.

Syntax *object*.GetNamedPersistedObject "name"

where:

object is an object expression that evaluates to a TipcMsg object.

name is the name of the user-defined object to retrieve.

Return Type VARIANT (IUnknown *) - A user-defined object

C API No comparable C API function

Example This example retrieves a persisted object of a named persisted object field of the message:

```
Dim out As Variant
Set out = msg.GetNamedPersistedObject("my_object")
Dim out2 As MyObject
Set out2 = out
```

GetNamedReal4 Method

The `GetNamedReal4` message retrieves the value of a named field in a message, which must be a 4-byte real number.

Syntax `object.GetNamedReal4 (name)`
where:
object is an object expression that evaluates to a `TipcMsg` object.
name is a string that specifies the field name to retrieve from a message.

Return Type `Single`

C API `TipcMsgGetNamedReal4`

Example This example retrieves the value of a named 4-byte real field of the message:

```
Dim r4 As Single
r4 = msg.GetNamedReal4("real4_data")
```

GetNamedReal4Array Method

The `GetNamedReal4Array` retrieves the value of a named field in a message, which must be an array of 4-byte real numbers.

Syntax `object.GetNamedReal4Array (name, real4_array_data, array_size)`
where:
object is an object expression that evaluates to a `TipcMsg` object.
name is a string that specifies the field name to retrieve from a message.
real4_array_data is an array of single real numbers used to hold the data retrieved from the message. This is passed as a `SAFEARRAY`.
array_size is a long integer that specifies the number of elements placed in *real4_array_data*.

Return Type `None`

C API `TipcMsgGetNamedReal4Array`

Example This example retrieves the value of a named 4-byte real array field of the message:

```
Dim r4_array(8) As Single
Dim numElements As Long
msg.GetNamedReal4Array("real4_array_data", r4_array, numElements)
```

GetNamedReal8 Method

The GetNamedReal8 method retrieves the value of a named field in a message, which must be an 8-byte real number. If necessary, the field is converted to the data format used by the process.

Syntax *object*.GetNamedReal8 (*name*)
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.

Return Type Double

C API TipcMsgGetNamedReal8

Example This example retrieves the value of a named 8-byte real field of the message:

Dim r8 As Double
r8 = msg.GetNamedReal8("real8_data")

GetNamedReal8Array Method

The GetNamedReal8Array method retrieves the value of a named field in a message, which must be an array of 8-byte real numbers.

Syntax *object*.GetNamedReal8Array (*name*, *real8_array_data*, *array_size*)
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.
real8_array_data is an array of double real numbers used to hold the data retrieved from the message. This is passed as a SAFEARRAY.
array_size is a long integer that specifies the number of elements placed in *real8_array_data*.

Return Type None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Remarks	The return value indicates how many actual items were placed into <i>real8_array_data</i> . A negative return value means that there are more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into <i>real8_array_data</i> . However, no data is placed into <i>real8_array_data</i> when this occurs.
C API	TipcMsgGetNamedReal8Array
Example	This example retrieves the value of a named 8-byte real array field of the message: <pre>Dim r8_array(16) As Double Dim numElements As Long msg.GetNamedReal8Array("r8_array_data", r8_array, numElements)</pre>

GetNamedReal16 Method

The GetNamedReal16 method retrieves the value of a named field in a message, which must be a 16-byte real number. If necessary, the field is converted to the data format used by the process.

Syntax	<i>object</i> .GetNamedReal16 (<i>name</i> , <i>real16_data</i>) where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>name</i> is a string that specifies the field name to retrieve from a message. <i>real16_data</i> is an array of 16 bytes to representing the 16 byte number.

Return Type None

C API TipcMsgGetNamedReal16

Example	This example retrieves the value of a named 16-byte real field of the message: <pre>Dim r16(16) As Byte msg.GetNamedReal16("real16_data", r16)</pre>

GetNamedReal16Array Method

The GetNamedReal16Array method retrieves the value of a named field in a message, which must be an array of 16-byte real numbers.

Syntax *object*.GetNamedReal16Array (*name*, *real16_array_data*, *array_size*)

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to retrieve from a message.

real16_array_data is an array of double real numbers used to hold the data retrieved from the message. This is passed as a SAFEARRAY.

array_size is a long integer that specifies the number of elements placed in *real16_array_data*.

Return Type None

C API TipcMsgGetNamedReal16Array

Example This example retrieves the value of a named 16-byte real array field of the message:

```
Dim r16_array(3)(16) As byte
Dim numElements As Long
msg.GetNamedReal16Array("r16_array_data", r16_array, numElements)
```

GetNamedStr Method

The GetNamedStr method retrieves the value of a named field, which must be a null-terminated character string field (field type STR). A string field is a special case of an array of characters. If necessary, the field is converted to the data format used by the process.

Syntax *object*.GetNamedStr (*name*)

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to retrieve from a message.

Return Type String

C API TipcMsgGetNamedStr

Example This example retrieves the value of a named string field of the message:

```
Dim s As String
s = msg.GetNamedStr("string_data")
```

GetNamedStrArray Method

The GetNamedStrArray method retrieves the value of a named field, which must be an array of null-terminated character string fields (field type STR). If necessary, the field is converted to the data format used by the process.

Syntax `object.GetNamedStrArray (name, str_array_data, array_size)`
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.
str_array_data is a string array used to hold the data retrieved from the message. This is passed as a SAFEARRAY.
array_size is a long integer that specifies the number of elements placed in *str_array_data*.

Return Type None

C API TipcMsgGetNamedStrArray

Example This example retrieves the value of a named string array field of the message:

```
Dim s_array(8) As String
Dim numElements As Long
msg.GetNamedStrArray("my_str", s_array, numElements)
```

GetNamedTimestamp Method

The GetNamedTimestamp method retrieves the value of a named field in a message, which must be a timestamp. If necessary, the field is converted to the data format used by the process. The timestamp is an eight-byte real number representing the current computer wall clock time in the number of seconds since midnight January 1, 1970.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Syntax `object.GetNamedTimestamp (name)`
 where:
object is an object expression that evaluates to a TipCMsg object.
name is a string that specifies the field name to retrieve from a message.

Return Type Double

C API TipCMsgGetNamedTimestamp

Example This example retrieves the value of a named timestamp field of the message:

```
Dim named_timestamp_return As Double
named_timestamp_return = msg.GetNamedTimestamp("timestamp_data")
```

GetNamedTimestampArray Method

The GetNamedTimestampArray method retrieves the value of a named field in a message, which must be an array of timestamps. A timestamp is an eight-byte real number representing the current computer wall clock time in the number of seconds since midnight January 1, 1970.

Syntax `object.GetNamedTimestampArray (name, timestamp_array_data, array_size)`
 where:
object is an object expression that evaluates to a TipCMsg object.
name is a string that specifies the field name to retrieve from a message.
timestamp_array_data is an array of double real numbers used to hold the data retrieved from the message. This is passed as a SAFEARRAY.
array_size is a long integer that specifies the number of elements placed in *real8_array_data*.

Return Type None

Remarks The return value indicates how many actual items were placed into *timestamp_array_data*. A negative return value means that there are more items to be placed into the array than there was space allocated. The negative number represents how many items would have been placed into *timestamp_array_data*. However, no data is placed into *timestamp_array_data* when this occurs.

C API TipCMsgGetNamedTimestampArray

Example This example retrieves the value of a named timestamp array field of the message:

```
Dim named_timestamp_array_return(16) As Double
Dim numElements As Long

msg.GetNamedTimestampArray("timestamp_array_data",
    named_timestamp_array_return, numElements)
```

GetNamedXml Method

The GetNamedXml method retrieves the value of a named field in a message, which must be a TipcXml message field.

Syntax *object*.GetNamedXml (*name*)
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to retrieve from a message.

Return Type TipcXml

C API TipcMsgGetNamedXml

Example This example retrieves the value of a named XML message field in the message:

```
Dim tXml As New TipcXml
Set tXml = msg.GetNamedXml("xml_data")
```

Load Method

The load method is inherited from IPersistStream; it provides a persistable interface to load a message from an IStream. This method is available for C++ only.

Syntax *object*->Load(*IStream* *PStm)
where:
object is an object expression that evaluates to an ITipcMsg pointer.
IStream is an IStream pointer to the stream from which the TipcMsg object is loaded.

Return Type None

C API No comparable C API function. See the Microsoft IPersistStream documentation.

Example This example Loads a message from an IStream:

```

LPPERSISTSTREAM ppersist;
LPSTREAM pstm;

ITipcMsgPtr pMsg;

blob = b;
size = s;
//allocate global memory
hmem = GlobalAlloc (GMEM_MOVEABLE, size);

//copy blob into global memory
memcpy (GlobalLock (hmem), blob, size);
GlobalUnlock (hmem);

//create stream around global memory
HRESULT status;
status = ::CreateStreamOnHGlobal (hmem, TRUE, &pstm);
hmem = 0;

//Get class ID from stream
CLSID clsID;
ULONG cbRead;
status = pstm->Read (&clsID, sizeof (CLSID), &cbRead);

//create class
status = ::CoCreateInstance (
    clsID,
    0,
    CLSCTX_ALL,
    IID_IUnknown,
    (void **) (pMsg));

//get the persist stream interface on TipcMsg object.
status = (*pMsg)->QueryInterface (&ppersist);

//load the object from stream
status = ppersist->Load (pstm);

// process data in the message

```

NextBinary Method

The NextBinary method retrieves the value of the current field of a message, which must be a binary field. If successful, it advances the current field to the next field in the message.

Syntax *object*.NextBinary (*binary_data*, *data_size*)
where:
object is an object expression that evaluates to a TipcMsg object.
binary_data is an array of bytes that specifies the binary data to retrieve from a message.
data_size is the number of bytes returned.

Return Type None

C API TipcMsgNextBinary

Example This example retrieves the value of the current binary field of a message:

```
Dim blob(110) As Byte
Dim numBytes as Long
msg.NextBinary blob, numBytes
```

NextChar Method

The NextChar method retrieves the value of the current field of a message, which must be a character field. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextChar
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type Integer

C API TipcMsgNextChar

Example This example retrieves the value of the current character field of the message:

```
Dim ch As Integer
ch = msg.NextChar
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

NextInt2 Method

The NextInt2 method retrieves the value of the current field of a message, which must be a 2-byte integer field. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextInt2

where *object* is an object expression that evaluates to a TipcMsg object.

Return Type Integer

C API TipcMsgNextInt2

Example This example retrieves the value of the current 2-byte integer field of the message:

```
Dim i2 As Integer
i2 = msg.NextInt2
```

NextInt2Array Method

The NextInt2Array method retrieves the value of the current field of a message, which must be an array of 2-byte integers. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextInt2Array (*int2_array_data*,*array_size*)

where:

object is an object expression that evaluates to a TipcMsg object.

int2_array_data is an array of 2-byte integers used to hold the data retrieved from the message. This is passed as a SAFEARRAY.

array_size is a long integer that specifies the number of elements placed in *int2_array_data*.

Return Type None

C API TipcMsgNextInt2Array

Example This example retrieves the value of the current 2-byte integer array field of the message:

```
Dim numElements As Long
Dim i2_array(12) As Integer

msg.NextInt2Array(i2_array, numElements)
```

NextInt4 Method

The NextInt4 method retrieves the value of the current field of a message, which must be a 4-byte integer field. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextInt4
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type Long (Integer)

C API TipcMsgNextInt4

Example This example retrieves the value of the current 4-byte integer field of the message:

```
Dim i4 As Long
i4 = msg.NextInt4
```

NextInt4Array Method

The NextInt4Array retrieves the value of the current field of a message, which must be an array of 4-byte integers. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextInt4Array (*int4_array_data*, *array_size*)
where:
object is an object expression that evaluates to a TipcMsg object.
int4_array_data is an array of long integers used to hold the data retrieved from the message. This is passed as a SAFEARRAY.

array_size is a long integer that specifies the number of elements placed in *int4_array_data*.

Return Type	None
C API	TipcMsgNextInt4Array
Example	This example retrieves the value of the current 4-byte integer array field of the message: <pre>Dim numElements As Long Dim i4_array(12) As Long msg.NextInt4Array(i4_array, numElements)</pre>

NextInt8 Method

The NextInt8 method retrieves the value of the current field of a message, which must be an 8-byte integer field. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax	<i>object</i>.NextInt8 where <i>object</i> is an object expression that evaluates to a TipcMsg object.
Return Type	Currency (CY)
C API	TipcMsgNextInt8
Example	This example retrieves the value of the current 8-byte integer field of the message: <pre>Dim i8 As Currency i8 = msg.NextInt8</pre>

NextInt8Array Method

The NextInt8Array retrieves the value of the current field of a message, which must be an array of 8-byte integers. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Syntax *object*.NextInt8Array (*int8_array_data*, *array_size*)

where:

object is an object expression that evaluates to a TipcMsg object.

int8_array_data is an array of Currency used to hold the data retrieved from the message. This is passed as a SAFEARRAY.

array_size is a long integer that specifies the number of elements placed in *int4_array_data*.

Return Type None

C API TipcMsgNextInt8Array

Example This example retrieves the value of the current 8-byte integer array field of the message:

```
Dim numElements As Long
Dim i8_array(12) As Currency
msg.NextInt8Array(i8_array, i8_array)
```

NextMsg Method

The NextMsg method retrieves the value of the current field of a message, which must be a message field. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextMsg

where *object* is an object expression that evaluates to a TipcMsg object.

Return Type TipcMsg

C API TipcMsgNextMsg

Example This example retrieves the value of the current message field of the message:

```
Dim msg2 As TipcMsg
Dim msgName As String
msg2 = msg.NextMsg
msgName = msg2.Type.Name
```


NextMsgArray Method

The NextMsgArray method retrieves the value of the current field of a message, which must be an array of messages. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax `object.NextMsgArray (msg_array_data, array_size)`

where:

object is an object expression that evaluates to a TipcMsg object.

msg_array_data is class TipcMsgArray, and is used to hold the data retrieved from the message. This is passed as a SAFEARRAY.

array_size is a long integer that specifies the number of elements placed in *msg_array_data*.

Return Type None

C API TipcMsgNextMsgArray

Example This example retrieves the value of the current message array field of the message:

```
Dim numElements As Long
Dim msg_array(8) As TipcMsg
Dim msgName As String
Dim i As Long

msg.NextMsgArray(msg_array, num_elements)
For i = 0 to numElements - 1
    msgName = msg_array(i).Type.Name
Next i
```

NextPersistedObject Method

The NextPersistedObject method returns a user-defined object.

Syntax `object.NextPersistedObject`

where *object* is an object expression that evaluates to a TipcMsg object.

Return Type VARIANT (IUnknown *) - A user-defined object

C API No comparable C API function

Example This example retrieves the value of the current persisted object field in the message:

```
Dim out As Variant
Set out = msg.NextPersistedObject
Dim out2 As MyObject
Set out2 = out
```

NextReal4 Method

The NextReal4 message retrieves the value of the current field of a message, which must be a 4-byte real number. If successful, it advances the current field to the next field in the message.

Syntax *object*.NextReal4
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type Single

C API TipcMsgNextReal4

Example This example retrieves the value of the current 4-byte real field of the message:

```
Dim r4 As Single
r4 = msg.NextReal4
```

NextReal4Array Method

The NextReal4Array retrieves the value of the current field of a message, which must be an array of 4-byte real numbers. If successful, it advances the current field to the next field in the message.

Syntax *object*.NextReal4Array (*real4_array_data*, *array_size*)
where:
object is an object expression that evaluates to a TipcMsg object.
real4_array_data is an array of single real numbers used to hold the data retrieved from the message. This is passed as a SAFEARRAY.
array_size is a long integer that specifies the number of elements placed in *real4_array_data*.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Return Type	None
C API	TipcMsgNextReal4Array
Example	This example retrieves the value of the current 4-byte real array field of the message: <pre>Dim r4_array(8) As Single Dim numElements As Long msg.NextReal4Array(r4_array, numElements)</pre>

NextReal8 Method

The NextReal8 method retrieves the value of the current field of a message, which must be an 8-byte real number. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax	<i>object</i>.NextReal8 where <i>object</i> is an object expression that evaluates to a TipcMsg object.
Return Type	Double
C API	TipcMsgNextReal8
Example	This example retrieves the value of the current 8-byte real field of the message: <pre>Dim r8 As Double r8 = msg.NextReal8</pre>

NextReal8Array Method

The NextReal8Array method retrieves the value of the current field of a message, which must be an array of 8-byte real numbers. If successful, it advances the current field to the next field in the message.

Syntax	<i>object</i>.NextReal8Array (<i>real8_array_data</i>, <i>array_size</i>) where: <i>object</i> is an object expression that evaluates to a TipcMsg object.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

real8_array_data is an array of 8-byte real numbers used to hold the data retrieved from the message. This is passed as a SAFEARRAY.

array_size is a long integer that specifies the number of elements placed in *real8_array_data*.

Return Type None

C API TipcMsgNextReal8Array

Example This example retrieves the value of the current 8-byte real array field of the message:

```
Dim r8_array(16) As Double
Dim numElements As Long

msg.NextReal8Array(r8_array, numElements)
```

NextReal16 Method

The NextReal16 method retrieves the value of the current field of a message, which must be a 16-byte real number. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextReal16
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type None

C API TipcMsgNextReal16

Example This example retrieves the value of the current 16-byte real field of the message:

```
Dim r16(16) As Byte

' fill in array
r16 = msg.NextReal16
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

NextReal16Array Method

The NextReal16Array method retrieves the value of the current field of a message, which must be an array of 16-byte real numbers. If successful, it advances the current field to the next field in the message.

Syntax	<i>object</i>.NextReal16Array (<i>real16_array_data</i>, <i>array_size</i>) where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>real16_array_data</i> is an array of double real numbers used to hold the data retrieved from the message. This is passed as a SAFEARRAY of 16 byte arrays. <i>array_size</i> is a long integer that specifies the number of elements placed in <i>real8_array_data</i>.
Return Type	None
C API	TipcMsgNextReal16Array
Example	This example retrieves the value of the current 16-byte real array field of the message: <pre>Dim 16_array(10)(16) As Byte Dim numElements As Long ' fill in array msg.NextReal16Array(r16_array, NumElements)</pre>

NextStr Method

The NextStr method retrieves the value of the current field, which must be a null-terminated character string field (field type STR). A string field is a special case of an array of characters. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax	<i>object</i>.NextStr where <i>object</i> is an object expression that evaluates to a TipcMsg object.
Return Type	String
C API	TipcMsgNextStr

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example retrieves the value of the current string field of the message:

```
Dim s As String
s = msg.NextStr
```

NextStrArray Method

The NextStrArray method retrieves the value of the current field, which must be an array of null-terminated character string fields (field type STR). If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax *object*.NextStrArray (*str_array_data*, *array_size*)
where:
object is an object expression that evaluates to a TipcMsg object.
str_array_data is a string array used to hold the data retrieved from the message. This is passed as a SAFEARRAY.
array_size is a long integer that specifies the number of elements placed in *str_array_data*.

Return Type None

C API TipcMsgNextStrArray

Example This example retrieves the value of the current string array field of the message:

```
Dim s_array(8) As String
Dim numElements As Long
msg.NextStrArray(s_array, numElements)
```

NextTimestamp Method

The NextTimestamp method retrieves the value of the current field of a message, which must be a timestamp. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process. The timestamp is an eight-byte real number representing the current computer wall clock time in the number of seconds since midnight January 1, 1970.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Syntax `object.NextTimestamp`
 where *object* is an object expression that evaluates to a TipcMsg object.

Return Type Double

C API TipcMsgNextTimestamp

Example This example retrieves the value of the current timestamp field of the message:

```
Dim timestamp_return As Double
timestamp_return = msg.NextTimestamp
```

NextTimestampArray Method

The NextTimestampArray method retrieves the value of the current field of a message, which must be an array of timestamps. If successful, it advances the current field to the next field in the message. The timestamp is an eight-byte real number representing the current computer wall clock time in the number of seconds since midnight January 1, 1970.

Syntax `object.NextTimestampArray (timestamp_array_data, array_size)`
 where:
object is an object expression that evaluates to a TipcMsg object.
timestamp_array_data is an array of timestamps used to hold the data retrieved from the message. This is passed as a SAFEARRAY.
array_size is a long integer that specifies the number of elements placed in *timestamp_array_data*.

Return Type None

C API TipcMsgNextTimestampArray

Example This example retrieves the value of the current timestamp array field of the message:

```
Dim timestamp_array(16) As Double
Dim numElements As Long
msg.NextTimestampArray(timestamp_array, numElements)
```

NextType Method

The NextType method retrieves the field type of the next field in a message.

Syntax *object*.NextType
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type TipcFt

C API TipcMsgNextType

Example This example retrieves the field type of the next field in a message:

```
Dim ft As TipcFt
ft = msg.NextType
```

NextUnknown Method

The NextUnknown method retrieves the value of the current field, which must have an unknown value, as set by the AppendUnknown Method.

Syntax *object*.NextUnknown
where *object* is an object expression that evaluates to a TipcMsg object.

Return Type None

C API TipcMsgNextUnknown

Example This example retrieves the value of the current character field of the message:

```
msg.nextUnknown
```

NextXml Method

The NextXml method retrieves the value of the current field of a message, which must be an Xml field. If successful, it advances the current field to the next field in the message. If necessary, the field is converted to the data format used by the process.

Syntax `object.NextXml`
 where *object* is an object expression that evaluates to a `TipcMsg` object.

Return Type `TipcXml`

C API `TipcMsgNextXml`

Example This example retrieves the value of the current message field of the message:

```
Dim xml As TipcXml
xml = msg.NextXml
```

Reset Method

The Reset method removes all fields and field data from a message.

Syntax `object.Reset`
 where *object* is an object expression that evaluates to a `TipcMsg` object.

Return Type `None`

C API `TipcMsgSetNumFields(msg, 0)`

Example This example clears the message of all fields and field data:

```
msg.Reset
```

Save Method

The Save method is inherited from `IPersistStream`; it provides a persistable interface to save a message from an `IStream`. This method is available for C++ only.

Syntax `Object->Save(IStream *PStm, BOOL fClearDirty)`
 where:
Object is an object expression that evaluates to an `ITipcMsg` pointer.
PStm is an `IStream` pointer to the stream into which the `TipcMsg` object is saved.

fClearDirty is a BOOL value indicating whether to clear the dirty flag after the save is complete.

Return Type	None
C API	No comparable C API function. See the Microsoft IPersistStream documentation.
Example	This example loads a message from an IStream: <pre>LPPERSISTSTREAM ppersist; LPSTREAM pstm; ITipcMsgPtr pMsg; .. create and save data to the message //get the persist stream interface on persistable object. status = pMsg->QueryInterface (&ppersist); //create a stream on HGlobal status = CreateStreamOnHGlobal (0, TRUE, &pstm); //get the class ID CLSID clsID; status = ppersist->GetClassID (&clsID); //save the class id to the stream ULONG cbWritten; status = pstm->Write (&clsID, sizeof (CLSID), &cbWritten); //save the object to the stream status = ppersist->Save (pstm, FALSE);</pre>

SetCompression Method

The SetCompression method sets the compression property of the message to True or False.

Syntax	<i>object</i>.SetCompression <i>compression</i> where: <i>object</i> is an object expression that evaluates to a TipcMsg object. <i>compression</i> is a Boolean value that specifies whether the message should be compressed: • True sets the message compression property to True. True causes the message to be compressed. • False sets the message compression property to False. False disables compression for message.

Return Type None

C API TipcMsgSetCompression

Example This example sets the compression property of a message:

```
Dim msgOut As TipcMsg
Dim compression As Boolean
Dim Tsvc As New TipcSvc

Set msgOut = Tsvc.MsgCreate(Tsvc.MtLookup("string_data"))
msgOut.Dest = "_all"

msgOut.SetCompression True
compression = msgOut.GetCompression
MsgBox(compression)
```

UpdateNamedBinary Method

The UpdateNamedBinary method updates the value of a binary field of a message using a name.

Syntax *object.UpdateNamedBinary (name, binary_data)*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to retrieve from a message.

binary_data is an array of bytes that specifies the binary data to retrieve from a message.

Return Type None

C API TipcMsgUpdateNamedBinary

Example This example retrieves the value of a named binary field in a message:

```
Dim blob(100) As Byte
' fill in blob
msg.UpdateNamedBinary("binary_data", blob)
```

UpdateNamedCharacter Method

The UpdateNamedChar method updates the value of a character field in a message, using a name. If necessary, the field is converted to the data format used by the process.

Syntax *object*.UpdateNamedCharacter *name*, *char*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
char is a character to replace the named field's data.

Return Type None

C API TipcMsgUpdateNamedChar

Example This example updates the value of a named character field in the message:

```
Dim ch As Byte
Ch = 64
msg.UpdateNamedChar("char_data", ch)
```

UpdateNamedInt2 Method

The UpdateNamedInt2 method updates a named 2-byte integer field (field type INT2) in a message.

Syntax *object*.UpdateNamedInt2 *name*, *int2_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in the message.
int2_data is the two byte integer data.

Return Type None

C API TipcMsgUpdateNamedInt2

Example This example appends a named 2-byte integer field in a message:

```
Dim i2 As Integer
i2 =33
msg.UpdateNamedInt2 "int2_data", i2
```

UpdateNamedInt2Array Method

The UpdateNamedInt2Array method updates a named 2-byte integer array field (field type INT2_ARRAY) in a message.

Syntax *object.UpdateNamedInt2Array name, int2_array_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to update.

int2_array_data is an integer array that specifies the array of 2-byte integers to update in message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgUpdateNamedInt2Array

Example This example appends a named 2-byte integer array field in a message:

```
Dim i2_array(16) As Integer
Dim i As Long
for i =0 to 15
i2_array(i) = i
Next i
msg.UpdateNamedInt2Array "int2Array_data", i2_array
```

UpdateNamedInt4 Method

The UpdateNamedInt4 method updates a named 4-byte integer field (field type INT4) in a message.

Syntax *object.UpdateNamedInt4 name, int4_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to update.

int4_data is a long integer that specifies the 4-byte integer data to update.

Return Type None

C API TipcMsgUpdateNamedInt4

Example This example appends a named 4-byte integer field in a message:

```
Dim i4 As Long
i4 = 123456789
msg.UpdateNamedInt4 "Int4_data", i4
```

UpdateNamedInt4Array Method

The UpdateNamedInt4Array method updates a named 4-byte integer array field (field type INT4_ARRAY) in a message.

Syntax *object.UpdateNamedInt4Array name, int4_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update.
int4_array_data is a long array that specifies the array of 4-byte integers to update in a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgUpdateNamedInt4Array

Example This example appends a named 4-byte integer array field in a message:

```
Dim i4_array(16) As Long
Dim i As Long
For i =0 to 15
i4_array(i) = i
Next i
msg.UpdateNamedInt4Array "Int4_Array_data", i4_array
```

UpdateNamedInt8 Method

The UpdateNamedInt8 method updates a named 8-byte integer field (field type INT8) in a message.

Syntax *object.UpdateNamedInt8 name, int8_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.

int8_data is a type of Currency (CY) that specifies the 8-byte integer data to update in a message.

Return Type None

C API TipcMsgUpdateNamedInt8

Example This example appends a named 8-byte integer field in a message:

```
Dim i8 As Currency
i8 = 123456789
msg.UpdateNamedInt8 "Int8_data", i8
```

UpdateNamedInt8Array Method

The UpdateNamedInt8Array method updates a named 8-byte integer array field (field type INT8_ARRAY) in a message.

Syntax *object.UpdateNamedInt8Array name, int8_array_data*

where:

object is an object expression that evaluates to a TipcMsg object.

name is a string that specifies the field name to update in a message.

int8_array_data is a long array that specifies the array of 8-byte integers to update in a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgUpdateNamedInt8Array

Example This example appends a named 8-byte integer array field in a message:

```
Dim i8_array(16) As Currency
Dim i As Long
For i = 0 to 15
    i8_array(i) = i
Next i
msg.UpdateNamedInt8Array "Int8_Array_data", i8_array
```

UpdateNamedMsg Method

The UpdateNamedMsg method updates a named message field (field type MSG) in a message.

Syntax *object.UpdateNamedMsg name, msg_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
msg_data is a TipcMsg object to replace the current TipcMsg object in the message.

Return Type None

C API TipcMsgUpdateNamedMsg

Example This example appends a named message field in a message:

```
Dim msg2 As TipcMsg
Dim mt As TipcMt
Set mt = TipcSvc1.MtLookup("control")
Set msg2 = TipcSvc1.MsgCreate(mt)
msg.UpdateNamedMsg "msg_data", msg2
```

UpdateNamedMsgArray Method

The UpdateNamedMsgArray method updates a named message array field (field type MSG_ARRAY) in a message.

Syntax *object.UpdateNamedMsgArray name, msg_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to updates in a message.
msg_array_data is a TipcMsgArray class that specifies the TipcMsg objects to update in the object message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgUpdateNamedMsgArray

Example This example appends a named message array field in a message:

```
Dim msg_array(4) As TipcMsg
Dim i As Long
Dim mt As TipcMt
Set mt = TipcSvc1.MtLookup("info")
For i = 0 to 3
Set msg_array(i) = TipcSvc1.MsgCreate(mt)
msg_array(i).AppendStr "Msg Number: " & Format(i)
Next i
msg.UpdateNamedMsgArray "msg_array_data", msg_array
```

UpdateNamedPersistedObject Method

The UpdateNamedPersistedObject method updates a user-defined object in a message. Internally this is stored as binary data in the message, but uses the IPersistStream interface to save the object to a blob first.

Syntax *object.UpdateNamedPersistedObject name, user_object*
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
user_object is a user object that has implemented the IPersistStream interface to save the object.

Return Type None

C API No comparable C API function

Example This example updates a named persisted object in a message:

```
Set contents = svc.MsgCreate(svc.MtLookup("info"))
contents.AppendStr "my string"
contents.Dest = "/mydest"
msg.UpdateNamedPersistedObject "fred", contents
```

UpdateNamedReal4 Method

The UpdateNamedReal4 method updates a named 4-byte real number field (field type REAL4) in a message.

Syntax *object.UpdateNamedReal4 name, real4_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
real4_data is a single real number that specifies the 4-byte real number to update in a message.

Return Type None

C API TipcMsgUpdateNamedReal4

Example This example updates a named 4-byte real field in a message:

```
Dim r4 As Single
r4 = 123.45
msg.UpdateNamedReal4 "real4_data", r4
```

UpdateNamedReal4Array Method

The UpdateNamedReal4Array method updates a named 4-byte real number array field (field type REAL4_ARRAY) in a message.

Syntax *object.UpdateNamedReal4Array name, real4_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
real4_array_data is a single array that specifies the array of 4-byte real numbers to update in a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgUpdateNamedReal4Array

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Example This example updates a 4-byte real array field in a message:

```
Dim r4_array(16) As Single
Dim i As Long
For i = 0 to 15
    r4_array(i) = 1234.123 + i
Next i
msg.UpdateNamedReal4Array "real4_array_data", r4_array
```

UpdateNamedReal8 Method

The UpdateNamedReal8 method updates a named 8-byte real number field (field type REAL8) in a message.

Syntax *object.UpdateNamedReal8 name, real8_data*

where:

object is an object expression that evaluates to a TipCMsg object.

name is a string that specifies the field name to update in a message.

real8_data is the 8-byte real number to update in a message.

Return Type None

C API TipCMsgUpdateNamedReal8

Example This example updates a named 8-byte real field in a message:

```
Dim r8 As Double
r8 = 1234567.123456
msg.UpdateNamedReal8 "real8_data", r8
```

UpdateNamedReal8Array Method

The UpdateNamedReal8Array method updates a named 8-byte real number array field (field type REAL8_ARRAY) in a message.

Syntax *object.UpdateNamedReal8Array name, real8_array_data*

where:

object is an object expression that evaluates to a TipCMsg object.

name is a string that specifies the field name to add in a message.

real8_array_data is a double array that specifies the array of 8-byte real numbers to update in a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgUpdateNamedReal8Array

Example This example updates an 8-byte real field in a message:

```
Dim r8_array(16) As Double
Dim i As Long
For i =0 to 15
r8_array(i) = 1234567.123456 + i
Next i
msg.UpdateNamedReal8Array "real8_array_data", r8_array
```

UpdateNamedReal16 Method

The UpdateNamedReal16 method updates a named 16-byte real number field (field type REAL16) in a message.

Syntax *object.UpdateNamedReal16 name, real16_data*
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
real16_data is the 16-byte real number to update in a message. This is implemented as an array of 16 bytes.

Return Type None

C API TipcMsgUpdateNamedReal16

Example This example updates a named 16-byte real field in a message:

```
Dim r16(16)As byte
' Fill in array
msg.UpdateNamedReal16 "real16_data", r16
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

UpdateNamedReal16Array Method

The UpdateNamedReal16Array method updates a named 16-byte real number array field (field type REAL16_ARRAY) in a message.

Syntax `object.UpdateNamedReal16Array name, real16_array_data`
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
real16_array_data is a double array that specifies the array of 16-byte real numbers to update in a message. This is passed as a SAFEARRAY, containing arrays of 16 bytes.

Return Type None

C API TipcMsgUpdateNamedReal16Array

Example This example updates a 16-byte real field in a message:

```
Dim r16_array(10)(16) As Byte
' fill in array here
msg.UpdateNamedReal16Array "real16_array_data", r16_array
```

UpdateNamedStr Method

The UpdateNamedStr method updates a named null-terminated character string field (field type STR) in a message. A string field is a special case of an array of characters.

Syntax `object.UpdateNamedStr name, str_data`
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
str_data is the string data to update in a message.

Return Type None

C API TipcMsgUpdateNamedStr

Example This example updates a string field in a message:

```
Dim s As String
s = "Test String")
msg.UpdateNamedStr "string_data", s
```

UpdateNamedStrArray Method

The UpdateNamedStrArray method updates a named string array field (field type STR_ARRAY) to the message.

Syntax *object.UpdateNamedStrArray name, str_array_data*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
str_array_data is a string array that specifies the array of string data to be added in a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgUpdateNamedStrArray

Example This example updates a string array field in a message:

```
Dim s_array(8) As String
Dim i As Long
For i =0 to 7
s_array(i) = "Test String" & Format(i)
Next i
msg.UpdateNamedStrArray "string_array_data", s_array
```

UpdateNamedTimestamp Method

The UpdateNamedTimestamp method updates a named timestamp field (field type TIMESTAMP) in a message. The timestamp is the current computer wall clock time in the number of seconds since midnight January 1, 1970. The TIMESTAMP field is an eight-byte real number containing *timestamp_data* to the end of a message’s data.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Syntax `object.UpdateNamedReal8 name, timestamp_data`
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
timestamp_data is the timestamp to update in a message.

Return Type None

C API TipcMsgUpdateNamedTimestamp

Example This example updates a named timestamp field in a message:

```
Dim timestamp As Double
timestamp = 1234567.123456
msg.UpdateNamedTimestamp "timestamp_data", timestamp
```

UpdateNamedTimestampArray Method

The UpdateNamedTimestampArray method updates a named timestamp array field (field type TIMESTAMP_ARRAY) in a message. The timestamp is the current computer wall clock time in the number of seconds since midnight January 1, 1970. The TIMESTAMP_ARRAY field is an eight-byte real number array containing *timestamp_array_data* to the end of a message's data.

Syntax `object.UpdateNamedTimestampArray name, timestamp_array_data`
 where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to add in a message.
timestamp_array_data is a double array that specifies the array of timestamps to update in a message. This is passed as a SAFEARRAY.

Return Type None

C API TipcMsgUpdateNamedTimestampArray

Example This example updates a named timestamp array field in a message:

```
Dim timestamp_array(16) As Double
Dim i As Long
For i = 0 to 15
  timestamp_array(i) = 1234567.123456 + i
Next i

msg.UpdateNamedTimestampArray "timestamp_array_data", timestamp_array
```

UpdateNamedXML Method

The UpdateNamedXml method updates a named TipcXml object (field type TIPCXML) in a message.

Syntax *object.UpdateNamedXml name, xml_object*
where:
object is an object expression that evaluates to a TipcMsg object.
name is a string that specifies the field name to update in a message.
xml_object is the TipcXml object to update in a message.

Return Type None

C API TipcMsgUpdateNamedXml

Example This example updates a named 4-byte integer field in a message:

```
Dim tXml As TipcXml
Dim tSvc as New TipcSvc
Set tXml = tSvc.XmlCreate("<data><data1><temp=hot/></data1></data>")
msg.UpdateNamedXml "Xml_data", tXml
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcMsg Class Properties

ArrivalTimestamp Property

The ArrivalTimestamp property returns the arrival timestamp of a message. This is a read-only property.

Syntax *object*.ArrivalTimestamp
where *object* is an object expression that evaluates to a TipcMsg object.

Data Type Double

C API TipcMsgGetArrivalTimestamp

Example This example gets the Arrival Timestamp from a message:

```
Dim timestamp as Double
timestamp = msg.ArrivalTimestamp
```

CorrelationID Property

The CorrelationID property returns the correlation ID of a message. This is a read-only property.

Syntax *object*.CorrelationID [*correlation_id*]
where:
object is an object expression that evaluates to a TipcMsg object.
correlation_id is the string representing the correlation ID to set in the message.

Data Type String

C API TipcMsgGetCorrelationId, TipcMsgSetCorrelationId

Example This example gets the CorrelationID from a message:

```
Dim corr_id as String
Msg.CorrelationID = "123-456-ABC"
Corr_id = Msg.CorrelationID
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

Current Property

The Current property sets or retrieves the current field of a message. This does not affect the number of fields in the message; new fields are still appended to the message.

Syntax *object*.Current[=*field_num*]
where:
object is an object expression that evaluates to a TipcMsg object.
field_num is a long integer that specifies the field number to use for the current field. Using 0 sets the field number to the first field.

Data Type Long (Integer)

C API TipcMsgSetCurrent, TipcMsgGetCurrent

Example This example sets the current field of the message:

```
Dim fieldNum As Long

'Determine field location in the message
fieldNum = msg.Current

'Point to the 3rd field of the message
msg.Current = 2

'Point to the very first field in the message
msg.Current = 0
```

CurrentUnknown Property

The CurrentUnknown property returns True if the value of the current field of a message is unknown or unset. This is a read-only property.

Syntax *object*.CurrentUnknown
where *object* is an object expression that evaluates to a TipcMsg object.

Data Type Boolean

C API TipcMsgGetCurrentFieldUnknown

Example This example determines if the current field of the message is of unknown value:

```
Dim fieldUnk As Boolean
fieldUnk = msg.CurrentUnknown
```

DeliveryMode Property

The DeliveryMode property specifies the level of guarantee, if any, provided when a message is sent. By default, messages are sent without guaranteed message delivery (GMD).

Syntax `object.DeliveryMode [=delivery_mode]`

where:

object is an object expression that evaluates to a TipcMsg object.

delivery_mode is an integer that specifies the delivery mode. Valid values are:

- 0 is for `tipcDeliveryBestEffort`, the default setting. When a message is sent, no special action is taken to ensure delivery of the message.
- 1 is for `tipcDeliverySome`, which is a GMD mode. The delivery of a message is successful when at least one of the specified receivers acknowledges delivery of the message.
- 2 is for `tipcDeliveryAll`, which is a GMD mode. The delivery of a message is successful when all specified receivers acknowledge delivery of the message.
- 3 is for `tipcDeliveryOrdered`. No special action is taken to ensure delivery of a message, but messages are received in the order they were published, even if a network or protocol failure occurs.

Data Type TipcDeliveryMode

Remarks For more information on delivery modes and GMD, see the *TIBCO SmartSockets User's Guide*.

C API TipcMsgSetDeliveryMode, TipcMsgGetDeliveryMode

Example This example sets the delivery mode of the message to `tipcDeliveryAll`:

```
Dim mode As Integer

'Determine delivery mode
mode = msg.DeliveryMode

'Change delivery mode to ensure that all get the message
msg.DeliveryMode = tipcDeliveryAll
```

DeliveryTimeout Property

The `DeliveryTimeout` property specifies the time period to wait for GMD (see `DeliveryMode` Property). The message `DeliveryTimeout` property is used to override a connection `DeliveryTimeout` property.

Syntax `object.DeliveryTimeout [=timeout]`
where:
object is an object expression that evaluates to a `TipcMsg` object.
timeout is a double real number that specifies the timeout period, in seconds, to wait for GMD delivery.

Data Type Double

Remarks See the GMD chapter in the *TIBCO SmartSockets User's Guide* for more information about delivery modes and delivery timeouts.

C API `TipcMsgSetDeliveryTimeout`, `TipcMsgGetDeliveryTimeout`

Example This example sets the delivery timeout of the message:

```
Dim timeout As Double

'Increase the current timeout by 10 seconds
timeout = msg.DeliveryTimeout
timeout = timeout + 10.0
msg.DeliveryTimeout = timeout
```

Dest Property

The Dest property specifies the destination of a message. The meaning of this property depends on how the message is used. Publish-subscribe messages between RTclients and RTservers use a subject name for the message destination and sender. Other applications might use this property for other purposes. The destination property default is set to an empty string ("").

Syntax `object.Dest [=destination]`
 where:
object is an object expression that evaluates to a TipcMsg object.
destination is a string that specifies the subject to which the message is published. It is user-defined in other applications.

Data Type String

C API TipcMsgSetDest, TipcMsgGetDest

Example This example sets the destination property of the message:

```
Dim d As String

'Determine current destination setting
d = msg.Dest

'Send message to /stocks/financial/... subject
msg.Dest = "/stocks/financial/..."
```

Fields Property

The Fields property is an instance of the TipcMsgFields class, containing the message's data fields. This is a read-only property, and need not be changed. To manipulate the fields within, see TipcMsgFields Class on page 377.

Syntax `object.Fields`
 where *object* is an object expression that evaluates to a TipcMsg object.

Data Type TipcMsgFields

C API No comparable C API function

Example This example prints out the type of each field in the message:

```
Dim field As Variant
For Each field in msg.Fields
    Select Case VarType(field)
        Case vbInteger
            Debug.Print "Int2 Field "
        Case vbInteger + vbArray
            Debug.Print "Int2Array Field "
        Case vbLong
            Debug.Print "Int4 Field "
        Case vbLong + vbArray
            Debug.Print "Int4Array Field "
        Case vbSingle
            Debug.Print "Real4 Field "
        Case vbSingle + vbArray
            Debug.Print "Real4Array Field "
        Case vbDouble
            Debug.Print "Real8 Field "
        Case vbDouble + vbArray
            Debug.Print "Real8Array Field "
        Case vbString
            Debug.Print "Str Field"
        Case vbString + vbArray
            Debug.Print "StrArray Field"
        Case Else
            Debug.Print "Unexpected Field type"
    End Select
Next field
```

HeaderStrEncode Property

The HeaderStrEncode property enables the compression of the message header into 4-byte integers for each message.

If the message header string encode is not set, the message type header string encode setting is used.

Syntax *object*.HeaderStrEncode [=header_str_encode]

where:

object is an object expression that evaluates to a TipcMsg object.

header_str_encode is a Boolean value:

- **True** enables the header string encode for each message. True indicates that this message's header is encoded.
- **False** disables the header string encode for each message, and the message type header string encode setting is used for this message. This is the default value for the header string encode property.

Data Type	Boolean
Remarks	For more information, see the <i>TIBCO SmartSockets User's Guide</i> .
C API	TipcMsgSetHeaderStrEncode, TipcMsgGetHeaderStrEncode
Example	This example sets the header string encoding property of the message: <pre>'Invert the HeaderStrEncode property of this message Dim encode As Boolean Dim opposite As Boolean encode = msg.HeaderStrEncode if encode Then opposite = False Else opposite = True End if msg.HeaderStrEncode = opposite</pre>

LbMode Property

In publish-subscribe operations, the (load balancing mode) property specifies to which RTclient a message is routed.

If the message load balancing mode is not set, the message type load balancing mode is used. The default value for the message type load balancing is `tipcLbNone`.

Syntax	<code>object.LbMode [=lb_mode]</code> where: <code>object</code> is an object expression that evaluates to a <code>TipcMsg</code> object. <code>lb_mode</code> is an integer that specifies the load balancing mode. Valid values are: • 0 is for <code>tipcLbNone</code>, the default setting. The message is routed with no load balancing. The message is routed to all RTclients subscribing to the destination subject. • 1 is for <code>tipcLbRoundRobin</code>. The message is routed to each RTclient subscribing to the destination subject, in turn. • 2 is for <code>tipcLbWeighted</code>. The message is routed to the RTclient with the fewest number of unacknowledged messages. • 3 is for <code>tipcLbSorted</code>. The message is routed to the RTclient in the lowest lexicographical order (sorted by unique subject).

Data Type	TipcLbMode

Remarks	See the <i>TIBCO SmartSockets User's Guide</i> for detailed information about load balancing.
C API	TipcMsgSetLbMode, TipcMsgGetLbMode
Example	This example changes a message's load balancing mode to round-robin: <pre>'Change Load Balancing Mode to Round Robin msg.LbMode = tipcLbRoundRobin</pre>

NumFields Property

The NumFields property sets the number of fields in message by truncating the message to the desired number of fields. If set to 0, the message is emptied.

Syntax *object*.NumFields [=*num_fields*]
where:
object is an object expression that evaluates to a TipcMsg object.
num_fields is a long integer that specifies how many fields to keep in the message.

Data Type Long (Integer)

C API TipcMsgGetNumFields, TipcMsgSetNumFields

Example This example retrieves and sets the number of fields in the message:

```
'Determine number of fields in a message
numFields = msg.NumFields 'OR
numFields = msg.Fields.Count

'Truncate message so that we can reuse it
msg.NumFields = 0
```


Priority Property

The Priority property sets the level of importance of a message. The greater the number specified, the higher the level of importance assigned to a message. When a message is sent, it is placed in the connection message queue of the receiving process in the order of priority (level of importance).

The message type priority is used, if set. If the message type priority is unknown, then the message priority default is `Default_Msg_Priority` which is 0.

Syntax `object.Priority [=priority]`

where:

`object` is an object expression that evaluates to a `TipcMsg` object.

`priority` is an integer that specifies the message priority level.

Data Type Integer

Remarks For more information on delivery modes and GMD, see the *TIBCO SmartSockets User's Guide*.

C API `TipcMsgSetPriority`, `TipcMsgGetPriority`

Example This example increases a message's priority by one:

```
Dim pri as Integer
pri = msg.Priority
pri = pri + 1
msg.Priority = pri
```

ReplyTo Property

The ReplyTo property returns the reply to property of the message. The ReplyTo property is used to indicate that the reply to a message is something other than the sender.

Syntax `object.ReplyTo [reply_to]`

where:

`object` is an object expression that evaluates to a `TipcMsg` object.

`reply_to` is the value to set the `reply_to` property of a message.

Data Type String

C API TipcMsgSetReplyTo, TipcMsgGetReplyTo

Example This example gets the ReplyTo property from a message:

```
Dim reply_to as String
Msg.ReplyTo = "/reply_subject"
Reply_to = msg.ReplyTo
```

Sender Property

The Sender property identifies the originator of a message. When the message is sent through publish-subscribe, the originator is the unique subject name of the sending process. The sender property is read-only.

Syntax *object*.Sender
where *object* is an object expression that evaluates to a TipcMsg object.

Data Type String

Remarks The Sender property cannot be set.

C API TipcMsgGetSender

Example This example determines the sender of a message:

```
Dim s As String
s = msg.Sender
```

SeqNum Property

The SeqNum property uniquely identifies the message for GMD so that duplicate messages can be detected by the receiver(s). When the message is sent with either the `tipcDeliverySome` or `tipcDeliveryAll` delivery modes, the sequence number of the message is set to a unique incremented number. Because the receiver records the sequence number when the message is received, if the message is later resent because of GMD failure, the receiver detects (and accepts) the resent message. The sequence number property is set automatically and cannot be set manually.

Syntax *object*.SeqNum
where *object* is an object expression that evaluates to a TipcMsg object.

Data Type	Long (Integer)
C API	TipcMsgSetSeqNum, TipcMsgGetSeqNum
Example	This example determines the sequence number of the message: <pre>Dim num As Long Dim msg As TipcMsg Set msg = TipcSvc1.Srv.MsgNext(2.0) if NOT msg is Nothing Then num = msg.SeqNum msg.Destroy End If</pre>

Type Property

The Type property identifies the kind of message being processed. This property must be supplied when creating a message.

Syntax	<code>object.Type [=mt]</code> where: <code>object</code> is an object expression that evaluates to a TipcMsg object. <code>mt</code> is class TipcMt, and specifies the predefined message type to which this message's format conforms.
Data Type	TipcMt
Remarks	See the information on message types in the <i>TIBCO SmartSockets User's Guide</i> for more information on the available message types.
C API	TipcMsgSetType, TipcMsgGetType
Example	This example changes the message type to numeric: <pre>Dim msgName as String msg.Type = TipcSvc1.MtLookup("numeric_data") msgName = msg.Type.Name</pre>

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

UserProp Property

The UserProp property is a user-defined property and can be used for any purpose, such as attaching a version number to messages. This property is not used internally by SmartSockets. The default value is 0.

Syntax *object*.UserProp [=*user_prop*]
where:
object is an object expression that evaluates to a TipcMsg object.
user_prop is a long integer. This is a user-defined value. The default value is 0.

Data Type Long (Integer)

C API TipcMsgSetUserProp, TipcMsgGetUserProp

Example This example adds 10 to the user property of a message:

```
Dim prop As Long  
  
prop = msg.UserProp  
prop = prop + 10  
msg.UserProp = prop
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

TipcMsgFields Class

TipcMsgFields — user class for message field operations

Description	The TipcMsgFields class provides an interface to the fields contained in SmartSockets messages. It allows you to add, retrieve and count the fields contained within messages. For more information on message fields, see the <i>TIBCO SmartSockets User's Guide</i> .
Methods	• Add Method • AppendUnknown Method
Properties	• Count Property • Item Property
Events	None

TipcMsgFields Class Methods

Add Method

The Add method appends data to a message, without specifying the data type. It is the simplest way to append data to a message, and can be used in place of the Append* or AddNamed* methods of the TipcMsg class. All the same data types that can be added with the Append* methods can be added with the Add method. As always, arrays must be one-dimensional.

Syntax *object*.Add *va_data* [, *name*]
 where:
object is an object expression that evaluates to a TipcMsgFields object.
va_data is a variant that specifies the data to add to the message. If the data is an array, the array must be one-dimensional.
name This is optional and used only if you want to add a named field. It is a string that specifies the field name to add to the message.

Return Type None

C API TipcMsgAppend*, TipcMsgAddNamed*

Example This example shows how to add each type of field to a message:

```
Dim nmsg As New TipcMsg
Dim nmt As New TipcMt
Dim b As Byte
Dim b_array(3) As Byte
Dim i2 As Integer
Dim i2_array(4) As Integer
Dim i4 As Long
Dim i4_array(2) As Long
Dim r4 As Single
Dim r4_array(2) As Single
Dim r8 As Double
Dim r8_array(2) As Double
Dim s As String
Dim s_array(2) As String

b_array(0) = Asc("a")
b_array(1) = Asc("B")
b = Asc("z")
i2 = 12
i2_array(0) = 98
i2_array(1) = 99
i4 = 12345678
i4_array(0) = 1000
i4_array(1) = 2000
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

```

r4 = 1234.123
r4_array(0) = 123.123
r4_array(1) = 567.789
r8 = 12345678.123456
r8_array(0) = 1000.34
r8_array(1) = 2000.345
s = "This is a test string"
s_array(0) = "Another test string"
s_array(1) = "And yet another"
nmt.Lookup "simple" 'This message type has grammar of int2
nmsg.Create nmt
nmsg.Fields.Add 23

msg.Fields.Add b_array 'AppendBinary
msg.Fields.Add b 'AppendChar
msg.Fields.Add i2 'AppendInt2
msg.Fields.Add i2_array 'AppendInt2Array
msg.Fields.Add i4 'AppendInt4
msg.Fields.Add i4_array 'AppendInt4Array
msg.Fields.Add r4 'AppendReal4
msg.Fields.Add r4_array 'AppendReal4Array
msg.Fields.Add r8 'AppendReal8
msg.Fields.Add r8_array 'AppendReal8Array
msg.Fields.Add s 'AppendStr
msg.Fields.Add s_array 'AppendStrArray
msg.Fields.Add nmsg 'AppendMsg
msg.Fields.Add nmsg "Named_Message" 'AddNamedMsg

```

AppendUnknown Method

The AppendUnknown method is similar to the Add method, except that the field type is explicitly given, and the value of that field is "no value". This is useful in cases where the data may be missing, for example, from a stream of telemetry data, but the fact that it is missing is important.

Syntax `object.AppendUnknown field_type`

where:

object is an object expression that evaluates to a TipcMsgFields object.

field_type is an integer that specifies the field type.

Return Type None

C API TipcMsgAppendUnknown

Example This example shows how to add a field with an unknown value to a message:

```
msg.Fields.AppendUnknown tipcFtChar 'char type
```

TipcMsgFields Class Properties

Count Property

The Count property returns the number of fields in the TipcMsgFields object. It is a read-only value.

Syntax *object*.Count
where *object* is an object expression that evaluates to a TipcMsgFields object.

Data Type Long (Integer)

C API TipcMsgGetNumFields

Example This example determines the number of fields in a message:

Dim nf As Long
nf = msg.Fields.Count

Item Property

The Item property returns the variant data in the specified field. The first field is specified by index 0.

Syntax *object*.Item (*index*)
where:
object is an object expression that evaluates to a TipcMsgFields object.
index is a long integer that specifies which field to return. A value of 0 specifies that the first field is returned.

Data Type Long (Integer)

Remarks The word "item" has special meaning in most ActiveX environments. The Item keyword can be omitted and *index* applied directly to the object. For fields with unknown values, those added with AddUnknown, the special variant type VT_EMPTY is returned.

C API No comparable C API function

Example This example illustrates the Item property used directly (A) and indirectly (B) to display the contents of a message's fields:

```
Dim msg as New TipcMsg

msg.Create (TipcSvc1.MtLookup("numeric_data"))
msg.Fields.Add 123.456
msg.Fields.Add 10.123
Debug.Print msg.Fields.Item(0)
' (A) debug window will display "123.456"
Debug.Print msg.Fields(1)
' (B) debug window will now also display "10.123"
```

TipcMt Class

TipcMt — user class for constructing, looking up and destroying message types

Description	The TipcMt class acts as an interface for creating message types. 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. For more information on message types, see the <i>TIBCO SmartSockets User's Guide</i> .
Methods	• Create Method • Destroy Method • GetCompression Method • Lookup Method • LookupByNum Method • SetCompression Method
Properties	• DeliveryMode Property • DeliveryTimeout Property • Grammar Property • HeaderStrEncode Property • LbMode Property • Name Property • Num Property • Priority Property • UserProp Property
Events	None

TipcMt Class Methods

Create Method

The Create method defines a new SmartSockets message type.

Syntax *object*.Create *name*, *num*, *grammar*
 where:
 object is an object expression that evaluates to a TipcMt object.
 name is a string representing the name of the message type. It is not
 case-sensitive.
 num is a long integer that specifies the numerical ID being assigned to the
 new message type.
 grammar is a string that specifies the description of the message's format used for
 reading and writing message files.

Return Type TipcMt

Remarks See the *TIBCO SmartSockets User's Guide* for more information about grammar and
 field types, as well as numeric message IDs. Typically, one would use
 TipcSvc::Create to create a message type. The create method is provided as a
 convenience to create many different message types from one TipcMt object.

C API TipcMtCreate

Example This example creates and then destroys a new message type test_mt:

```
Dim mt as New TipcMt
mt.Create "test_mt", 3000, "str int4 int4 msg"
mt.Destroy
```

Destroy Method

The Destroy method destroys a message type. When a message type is destroyed, all memory for the message type is freed. This also allows you to create new message types with the same numeric assignment in the place of the destroyed message type.

Syntax	<i>object</i> .Destroy where <i>object</i> is an object expression that evaluates to a TipcMt object.
Return Type	None
Remarks	It is possible to destroy SmartSockets predefined message types.
C API	TipcMtDestroy
Example	This example creates multiple message types with the same number, then destroys the message types: <pre>Dim mt As TipcMt Dim svc as new TipcSvc Dim msg as TipcMsg Set mt = svc.MtCreate "test_mt", 3000, "str int4 int4" mt.Create "test_mt2:", 3001, "str str str" set msg = TipcSvc1.MsgCreate(mt) msg.Fields.Add "Test string" msg.Fields.Add 99999 msg.Fields.Add 12345 msg.Dest "tmp" TipcSvc1.Srv.MsgSend msg TipcSvc1.Srv.Flush 'Destroy the message type so that it can be reused mt.Destroy mt.Create "another mt", 3000, "int2 int2_array str" set msg = TipcSvc1.MsgCreate(mt)</pre>

GetCompression Method

The GetCompression method returns the compression property of the message type.

Syntax	<i>object</i> .GetCompression where <i>object</i> is an object expression that evaluates to a TipcMt object.
Return Type	Boolean
C API	TipcMtGetCompression

Example This example gets the compression property of a message type:

```
Dim mtHi As TipcMt
Dim compression As Boolean
Dim Tsvc As New TipcSvc

Set mtHi = Tsvc.MtCreate("hi", 1, "str str int2")
mtHi.SetCompression True
compression = mtHi.GetCompression
MsgBox (compression)
```

Lookup Method

The Lookup method initializes a TipcMt object to be a SmartSockets message of the specified type. The predefined message type is specified by its descriptive name.

Syntax *object.Lookup name*

where:

object is an object expression that evaluates to a TipcMt object.

name is a string representing the message type name to look up. It is not case-sensitive.

Return Type None

Remarks Typically, TipcSvc::MtLookup should be used instead of this call.

C API TipcMtLookup

Example This example looks up the control and Info message types:

```
Dim mt As TipcMt
Dim svc as New TipcSvc

Set mt = svc.MtLookup("control")
Debug.Print mt.Name
mt.Lookup "Info"
Debug.Print mt.Name
```

LookupByNum Method

The `LookupByNum` method initializes a `TipcMt` object to be a `SmartSockets` message of the specified type. The predefined message type is specified by its numeric assignment.

Syntax `object.LookupByNum num`
where:
object is an object expression that evaluates to a `TipcMt` object.
num is a long integer that specifies the numeric ID of the `SmartSockets` message type to look up.

Return Type None

Remarks Typically, `TipcSvc::MtLookupByNum` should be used instead of this call.

C API `TipcMtLookupByNum`

Example This example looks up message types by their associated number:

```
Dim mt As New TipcMt
mt.LookupByNum -30 'Control Message type
mt.LookupByNum -40 'Info Message Type
```

SetCompression Method

The `SetCompression` method sets the compression property of the message type to `True` or `False`. By default, messages use the message type's compression property.

Syntax `object.SetCompression compression`
where:
object is an object expression that evaluates to a `TipcMt` object.
compression is a Boolean value that specifies whether a message of this message type should be compressed:

- `True` sets the message type compression property to `True`. `True` causes the message to be compressed.
- `False` sets the message type compression property to `False`. `False` disables compression for message.

Return Type None

C API TipcMtSetCompression

Example This example sets the compression property of a message type:

```
Dim mtHi As TipcMt
Dim compression As Boolean
Dim Tsvc As New TipcSvc

Set mtHi = Tsvc.MtCreate("hi", 1, "str str int2")
mtHi.SetCompression True
compression = mtHi.GetCompression
MsgBox (compression)
```

TipcMt Class Properties

DeliveryMode Property

The `DeliveryMode` property specifies the default level of guarantee, if any, when a message of the `TipcMt` object's type is sent. By default, messages are sent without GMD.

Syntax `object.DeliveryMode [=default_mode]`
where:
`object` is an object expression that evaluates to a `TipcMt` object.
`default_mode` is a `TipcDeliveryMode` class that specifies the default delivery mode for all messages of this message type. Valid values are:

- 0 is for `tipcDeliveryBestEffort`, the default setting. When a message is sent, no special action is taken to ensure delivery of the message.
- 1 is for `tipcDeliverySome`, which is a GMD mode. The delivery of a message is successful when at least one of the specified receivers acknowledges delivery of the message.
- 2 is for `tipcDeliveryAll`, which is a GMD mode. The delivery of a message is successful when all specified receivers acknowledge delivery of the message.
- 3 is for `tipcDeliveryOrdered`. No special action is taken to ensure delivery of a message, but messages are received in the order they were published, even if a network or protocol failure occurs.

Data Type	<code>TipcDeliveryMode</code>
Remarks	For more information on delivery modes, see the GMD chapter in the <i>TIBCO SmartSockets User's Guide</i> .
C API	<code>TipcMsgSetDeliveryMode</code> , <code>TipcMsgGetDeliveryMode</code>

Example This example creates a new message type that is sent with GMD:

```
Dim mt As TipcMt
Dim mode As Integer

mt.Create "test_mt", 3000, "int4 int4"
mt.DeliveryMode = tipcDeliveryAll
mode = mt.DeliveryMode
If mode <> tipcDeliveryAll Then
    Debug.Print "Unable to set Delivery Mode successfully"
End If
```

DeliveryTimeout Property

The DeliveryTimeout property identifies the time period to wait for GMD (see DeliveryMode Property). The message DeliveryTimeout property is used to override a connection DeliveryTimeout property.

Syntax *object*.DeliveryTimeout [=timeout]

where:

object is an object expression that evaluates to a TipcMt object.

timeout is a double real number that specifies the time, in seconds, to wait for GMD messages to be delivered.

Data Type Double

Remarks For more information, see the *TIBCO SmartSockets User's Guide*.

C API TipcMtSetDeliveryTimeout, TipcMtGetDeliveryTimeout

Example This example increases the delivery timeout for a message type by 10 seconds:

```
Dim mt As New TipcMt
Dim timeout As Double

mt.Create "new message", 1000, "int4 real8"
timeout = mt.DeliveryTimeout
timeout = timeout + 10
mt.DeliveryTimeout = timeout
```

Grammar Property

The Grammar property identifies the format for writing out the contents of a message to message files. The Grammar property is read-only.

Syntax `object.Grammar`
where *object* is an object expression that evaluates to a TipcMt object.

Data Type String

Remarks See the *TIBCO SmartSockets User's Guide* for information about grammar and message types.

C API TipcMtGetGrammar

Example This example retrieves the grammar of a monitoring message:

```
Dim mt as New TipcMt
Dim s As String
Dim n As String

mt.Lookup "mon_server_general_buffer_poll_result"
s = mt.Grammar
n = mt.Name
Debug.Print "Grammar of " & n & " is <" & s & ">"
```

HeaderStrEncode Property

The HeaderStrEncode property specifies whether or not the compression of the message header into 4-byte integers happens by default for this message type.

Syntax `object.HeaderStrEncode [=header_str_encode]`
where:
object is an object expression that evaluates to a TipcMt object.
header_str_encode is a Boolean value that specifies the default message header compression mode for messages of this type:

- True means that by default, message headers are compressed.
- False means that message headers are not compressed. This is the default value for the header string encode property.

Data Type Boolean

- Remarks** Setting the header string encode for each message can be useful, but setting the message type header string encode property changes the default behavior of all outgoing messages of that specific type. For more information, see the *TIBCO SmartSockets User's Guide*.
- C API** TipcMtSetHeaderStrEncode, TipcMtGetHeaderStrEncode
- Example** This example displays text in the Debug window to indicate if header encoding is turned on by default for the message type used by the msg message:
- ```

mst = msg.Type.Name
If (msg.Type.HeaderStrEncode = True) Then
 Debug.Print "Encoding is On for type ",mst;
Else
 Debug.Print "Encoding is Off for type ",mst;
End If

```

## LbMode Property

---

The LbMode property identifies to which RTclient a message is routed, depending on the load balancing mode specified. This value is used as the default load balancing mode for messages of this type.

- Syntax** `object.LbMode [=default_lb_mode]`
- where:
- `object` is an object expression that evaluates to a TipcMt object.

*default\_lb\_mode* is class `TipcLbMode` and specifies the default load balancing mode for messages of this message type. Valid values are:

- `tipcLbNone`  
means the message is routed with no load balancing. The message is routed to all RTclients subscribing to the destination subject.
- `tipcLbRoundRobin`  
means the message is routed to each RTclient subscribing to the destination subject, in turn.
- `tipcLbSorted`  
means the message is routed to the RTclient in the lowest lexicographical order (sorted by unique subject).
- `tipcLbWeighted`  
means the message is routed to the RTclient with the fewest number of unacknowledged messages.

|           |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Data Type | TipcLbMode                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Remarks   | See the <i>TIBCO SmartSockets User's Guide</i> for detailed information about load balancing.                                                                                                                                                                                                                                                                                                                                               |
| C API     | TipcMtSetLbMode, TipcMtGetLbMode                                                                                                                                                                                                                                                                                                                                                                                                            |
| Example   | <div>This example prints out the load balancing mode:<pre>mst = msg.Type.Name Select Case msg.Type.LbMode Case tipcLbNone     Debug.Print "No load balancing for " &amp; mst Case tipcLbRoundRobin     Debug.Print "Round Robin load balancing for " &amp; mst Case tipcLbSorted     Debug.Print "Sorted load balancing for " &amp; mst Case tipcLbWeighted     Debug.Print "Weighted load balancing for " &amp; mst End Select</pre></div> |

## Name Property

---

The Name property returns the name of the message type specified by the object. It is a read-only value and is set when the message type is created.

**Syntax**     `object.Name`

where *object* is an object expression that evaluates to a TpcMt object.

**Data Type**     String

**C API**     TpcMtGetName

**Example**     This example displays the name of the type of message:

```
Debug.Print "The variable 'msg' contains a message
 of type "; msg.Type.Name
```

## Num Property

---

The Num property returns the assigned numeric ID of the message type specified by the object. It is a read-only value and is set when the message type is created.

**Syntax**     `object.Num`

where *object* is an object expression that evaluates to a TpcMt object.

**Data Type**     Long (Integer)

**C API**     TpcMtGetNum

**Example**     This example displays the identifying number of the message type:

```
Debug.Print "The variable 'msg' contains a message
 with type ID="; msg.Type.Num
```

## Priority Property

The Priority property identifies the default level of importance for messages of this message type.

**Syntax**     *object*.Priority [=priority]  
              where:  
              *object*     is an object expression that evaluates to a TipcMt object.  
              *priority*    is an integer that specifies the default priority for messages of this message type.

**Data Type**     Integer

**Remarks**     The greater the number specified, the higher the level of importance assigned to a message. When a message is sent, it is placed in the connection message queue of the receiving process in the order of priority. If the message type priority is not set, then the message priority default is Default\_Msg\_Priority, which is 0.

**C API**     TipcMtSetPriority, TipcMtGetPriority

**Example**     This example sets the priority property of the message type:

```
Dim mt as New TipcMt
mt.Create("priority_test",2002,"int4")
mt.Priority = 5
Debug.Print "Newly created message type's priority is";
mt.Priority
mt.Destroy
```

## UserProp Property

The UserProp is a user-defined property and can be used for any purpose, such as attaching a version number to messages. This property is not used internally by SmartSockets. The user property is 0 by default.

**Syntax**     *object*.UserProp [=user\_prop\_value]  
              where:  
              *object*             is an object expression that evaluates to a TipcMt object.  
              *user\_prop\_value*    is a long integer that specifies the default user-defined property for messages of this message type.

**Data Type**     Long (Integer)

**C API**    TipcMtSetUserProp, TipcMtGetUserProp

**Example**    This example sets the user property of a new message type:

```
Dim mt As New TipcMt
Dim prop as Long

mt.Create "new message", 1000, "int4"
mt.UserProp = 100
prop = mt.UserProp
If prop <> 100 Then
 Debug.Print "Unable to set User Property"
End If
```

## TipcProcessCb Class

TipcProcessCb — class for handling SmartSockets messages arriving to your program from the TipcMsg event

**Description** The TipcProcessCb component contains the TipcProcessCb class, providing the TipcProcessCb event that invokes when a SmartSockets message is delivered to your program. The TipcProcessCb object must first be activated by adding it to a TipcConn or TipcSrvConn object. See the AddProcessCb and AddDefaultCb methods of those classes for more information. For more information on callbacks, see the *TIBCO SmartSockets User's Guide*.

Users must implement their own class from TipcProcessCb using the implements keyword in a separate class module.

For example, `' class module my process callback implements TipcProcessCb.`

**Methods** None

**Properties** None

**Events** TipcMsg Event

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED



## TipcProcessCb Class Events

### TipcMsg Event

The TipcMsg event is invoked when a SmartSockets message is delivered to an RTclnt. Place message-handling code in this event.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <div> Private Sub TipcProcessCb_TipcMsg (<i>conn</i>, <i>msg</i>, <i>param</i>)</div> <div> where:</div> <div> <i>conn</i>   Currently reserved.</div> <div> <i>msg</i>    is class TipcMsg and is the actual SmartSockets message that is received.</div> <div> <i>param</i> is optional and is an <i>ole_variant</i> that specifies a user-defined argument set when registering the callback. See TipcSrvConn.AddProcessCb, or TipcSrvConn.AddSubjectCb, or TipcSrvConn.AddDefaultCb.</div>                                           |
| <b>Return Type</b> | None                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Remarks</b>     | You must register a user-created class implemented from TipcProcessCb with the AddDefaultCb, AddProcessCb, AddSubjectCb or AddSubjectDefaultCb methods for the TipcMsg event to be invoked.                                                                                                                                                                                                                                                                                                                                              |
| <b>C API</b>       | Comparable to user-defined functions installed with TipcSrvConnProcessCbCreate, TipcSrvConnDefaultCbCreate, TipcSrvConnSubjectCbCreate and TipcSrvConnSubjectDefaultCbCreate.                                                                                                                                                                                                                                                                                                                                                            |
| <b>Example</b>     | <div>This example displays every field of every message it receives:</div> <pre> Private Sub TipcProcessCb_TipcMsg(ByVal conn As Object, ByVal pMsg As TIPCCOMLib.ITipcMsg, ByVal arg As Variant) Dim tmsg as TipcMsg Dim nmsg As TipcMsg Dim field As Variant Dim i As Long  Set tmsg = pMsg For Each field in tmsg     if VarType(field) &gt; vbArray Then         Debug.Print TypeName(field) &amp; " received"         For i = LBound(field) to UBound(field) - 1             Debug.Print Format(field(i))         Next i     </pre> |

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

```
Else
 If VarType(field) = vbObject Then
 set nmsg = field
 Debug.Print nmsg.Type.Name & " message received"
 Else
 Debug.Print TypeName(field) & ": " & Format(field)
 End If
End If
Next field
End Sub
```

## TipcSrvConn Class

TipcSrvConn — class for managing a connection to RTserver and messaging tasks your application performs once connected to an RTserver

**Description** The TipcSrvConn class provides an interface to the function to send, receive and process SmartSockets messages, as well as manage RTclient's publish-subscribe participation. For more information on RTserver, see the *TIBCO SmartSockets User's Guide*. There may be several instances of TipcSrvConn objects created, each representing a connection to a RTserver.

### Methods

- CbDestroy Method
- Close Method
- Create Method
- DefaultCbCreate Method
- ErrorCbCreate Method
- GetBufferReadSize Method
- GetSubscribe Method
- GetTrafficBytesRecv Method
- GetTrafficMsgsRecv Method
- GmdFileCreate Method
- GmdMsgDelete Method
- GmdMsgStatus Method
- MainLoop Method
- MsgNext Method
- MsgSearchType Method
- MsgSendRpc Method
- OpenCbCreate Method
- SetCredentials Method
- SetUsernamePassword Method
- SrvLock Method
- TipcSrvMonExtCreate Method
- SubjectCbCreate Method
- SubjectDefaultCbCreate Method
- SubjectLbInit Method
- Check Method
- CloseCbCreate Method
- CreateNamed Method
- Destroy Method
- Flush Method
- GetBufferWriteSize Method
- GetSubscribeLb Method
- GetTrafficBytesSent Method
- GetTrafficMsgsSent Method
- GmdFileDelete Method
- GmdMsgServerDelete Method
- KeepAlive Method
- MsgInsert Method
- MsgProcess Method
- MsgSend Method
- Open Method
- ProcessCbCreate Method
- SetStdSubjectSubscribe Method
- SrvIsRunning Method
- TipcSrvMonCreate Method
- SrvUnlock Method
- SubjectCbDestroyAll Method
- SubjectGmdInit Method
- SubjectSetSubscribe Method

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

- SubjectSetSubscribeEx Method
- SubjectSetSubscribeLb Method

Properties

- Arch Property
- BlockMode Property
- DefaultSubjectPrefix Property
- GmdNumPending Property
- Node Property
- Pid Property
- ServerAutoConnect Property
- ServerDeliveryTimeout Property
- ServerMaxReconnectDelay Property
- ServerNames Property
- ServerStartDelay Property
- ServerStartMaxTries Property
- Status Property
- Timeout Property
- User Property
- AutoFlushSize Property
- DefaultProtocols Property
- GmdMaxSize Property
- MonitorScope Property
- NumQueued Property
- Project Property
- ServerAutoFlushSize Property
- ServerKeepAliveTimeout Property
- ServerMsgSend Property
- ServerReadTimeout Property
- ServerStartMaxTimeout Property
- ServerWriteTimeout Property
- Subjects Property
- UniqueSubject Property
- XtSource Property

Events      None

## TipcSrvConn Class Methods

### CbDestroy Method

The CbDestroy method removes a callback from a TipcSrvConn object.

**Syntax**     *object.CbDestroy cb\_tag*  
 where:  
*object*     is an object expression that evaluates to a TipcSrvConn object.  
*cb\_tag*     Is a Long 4 byte integer that represents the callback. This is retrieved in a callback create call.

**Return Type**     None

**C API**     TutCbDestroy

**Example**     This example creates then destroys an open callback:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim cb_obj as new MyCallBackObject
Dim cb_tag as long

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
cb_tag = SrvConn.CloseCbCreate cb_obj, 1, "argument"
SrvConn.Open
SrvConn.CbDestroy cb_tag
SrvConn.Close
```

### Check Method

The Check method checks to see if data can be read from or written to the connection to an RTserver. This is a low-level function that typically is not used directly by developers.

**Syntax**     *object.Check (check\_mode, timeout)*  
 where:  
*object*     is an object expression that evaluates to a TipcSrvConn object.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

*check\_mode* is class `TipcCheckMode`, and specifies the mode to check for availability:

- 0 is `tipcCheckRead` and means check for data read availability
- 1 is `tipcCheckWrite` and means check for data write availability

*timeout* is a double real number that specifies how long, in seconds, to wait before returning a `False` value.

|             |                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Return Type | Boolean, where:<br>True indicates data can be read from or written to the connection to an RTserver.<br>False indicates data cannot be read from or written to an RTserver within <i>timeout</i> seconds. |
| C API       | TipcSrvConnCheck                                                                                                                                                                                          |
| Example     | This example checks to see if data is available for reading from the RTserver in three seconds:<br><br>If (Not SrvConn.Check(TipcCheckRead, 3.0)) Then<br>Debug.Print "Can't read!"<br>End If             |

## Close Method

The `Close` method closes a connection to RTserver. Any close callbacks will be called at this time.

|             |                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <i>object</i> .Close<br>where <i>object</i> is an object expression that evaluates to a <code>TipcSrvConn</code> object.                                                                                                 |
| Return Type | None                                                                                                                                                                                                                     |
| C API       | TipcSrvConnClose                                                                                                                                                                                                         |
| Example     | This example creates, opens, and closes a <code>TipcSrvConn</code> :<br><br>Dim SrvConn as TipcSrvConn<br>Dim Svc as new TipcSvc<br>Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")<br>SrvConn.Open<br>SrvConn.Close |

## CloseCbCreate Method

The CloseCbCreate method registers a user-defined callback to be called when a connection is closed. Multiple callbacks may be registered. See TipcConnChangeCb.

**Syntax** *object*.CloseCbCreate *event\_handler*, *priority*, *arg*

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*event\_handler* Is a user-defined callback object, implemented from the TipcConnChangeCb class.

*priority* Is a long representing the priority of the object.

*arg* Is a variant user argument that is passed to the ConnChange event in the user-defined callback class implemented from TipcConnChangeCb.

**Return Type** Long (callback tag used to remove a callback if required)

**C API** TipcSrvConnCloseCbCreate

**Example** This example creates a close callback, which is called during the conn.close method:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim cb_obj as new MyCallBackObject
Dim cb_tag as long

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
cb_tag = SrvConn.CloseCbCreate cb_obj, 1, "argument"
SrvConn.Open
SrvConn.Close
```

## Create Method

The Create method creates a connection to an RTserver. See the TipcSvc TipcSrvConnCreate Method.

**Syntax** *object*.Create

where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** TipcSrvConn

**C API**     TipcSrvConnCreate

**Example**     See the TipcSvc TipcSrvConnCreate Method.

## CreateNamed Method

The CreateNamed method creates a named connection to an RTserver. See the TipcSvc TipcSrvConnCreateNamed Method.

**Syntax**     *object*.CreateNamed  
              where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type**     TipcSrvConn

**C API**     TipcSrvConnCreateNamed

**Example**     See the TipcSvc TipcSrvConnCreateNamed Method.

## DefaultCbCreate Method

The DefaultCbCreate method installs an event handler for SmartSockets messages that arrive at RTclient. The default callback is used when there is no process callback for the particular type of message being received.

**Syntax**     *object*.DefaultCbCreate *event\_handler*,*priority*, *param*  
              where:  
              *object*        is an object expression that evaluates to a TipcSrvConn object.  
              *event\_handler* evaluates to a user-defined class implemented from the TipcProcessCb object whose TipcMsg event is invoked when a message with no other applicable handler is received.  
              *priority*       is an integer that specifies the priority of this handler. The higher priority handlers of the same type are called first.  
              *param*        is an ole\_variant that specifies a user-defined argument accessible when a callback function is invoked. This parameter is useful to avoid redundant code. For example, if updating several cells in a spread sheet using the same callback function, this parameter could be used as the cell to update.



|             |                                                                                                                                                                                                                                                                                                                        |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Return Type | None                                                                                                                                                                                                                                                                                                                   |
| Remarks     | See the <i>TIBCO SmartSockets User's Guide</i> for more information on default callbacks. Creating a default callback opens a connection if it is not currently open.                                                                                                                                                  |
| C API       | TipcSrvDefaultCbCreate                                                                                                                                                                                                                                                                                                 |
| Example     | <p>This example installs a default event handler:</p> <pre>Dim SrvConn as TipcSrvConn Dim Svc as new TipcSvc Dim cb_obj as new MyDefaultCallBackObject Dim cb_tag as long  Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "") SrvConn.Open cb_tag = SrvConn.DefaultCbCreate cb_obj, 1, "argument" SrvConn.Close</pre> |

## Destroy Method

The Destroy method destroys a TipcSrvConn object along with the connection to an RTserver if open.

|             |                                                                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p><i>object</i>.Destroy</p> <p>where <i>object</i> is an object expression that evaluates to a TipcSrvConn object.</p>                                                               |
| Return Type | None                                                                                                                                                                                  |
| C API       | TipcSrvConnDestroy                                                                                                                                                                    |
| Example     | <p>This example clears out a TipcSrvConn object:</p> <pre>Dim SrvConn as TipcSrvConn Dim svc as new TipcSvc Set SrvConn = svc.TipcSrvConnCreate("", "", "", "") SrvConn.Destroy</pre> |

## ErrorCbCreate Method

The ErrorCbCreate method installs an event handler for SmartSocket errors occurring with the connection encapsulated by the TipcSrvConn object.

**Syntax**     *object*.ErrorCbCreate *event\_handler*, *priority*, *param*

where:

*object*             is an object expression that evaluates to a TipcSrvConn object.

*event\_handler*     evaluates to a user-defined class implemented from the TipcErrorCb object whose TipcMsg event is invoked when a message with no other applicable handler is received.

*priority*           is an integer that specifies the priority of this handler. The higher priority handlers of the same type are called first.

*param*             is an *ole\_variant* that specifies a user-defined argument accessible when a callback function is invoked. This parameter is useful to avoid redundant code. For example, if updating several cells in a spread sheet using the same callback function, *param* could be the cell to update.

**Return Type**     None

**Remarks**     See the *TIBCO SmartSockets User's Guide* for more information on default callbacks. Creating an error callback opens a connection if it is not currently open. To remove or restore the default error callback, which attempts to reconnect to RTserver, use the RemoveDefaultErrorCb and RestoreDefaultErrorCb. To ensure a user-defined callback is called before the default error callback, use a priority of at least 1.

**C API**     TipcSrvErrorCbCreate

**Example**     This example installs an error callback handler:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim cb_obj as new MyErrorCallBackObject
Dim cb_tag as long

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
cb_tag = SrvConn.ErrorCbCreate cb_obj, 1, "argument"
...
SrvConn.Close
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## Flush Method

The Flush method flushes the buffered outgoing messages on the connection to an RTserver.

**Syntax** *object.Flush*

where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** None

**Remarks** Flushing buffered messages can significantly influence performance of an application. While there are occasions to flush immediately after a message is sent, if this is done after every message, network bandwidth is wasted and throughput is diminished dramatically. Rather than explicitly flushing messages on a regular basis, use the AutoFlushSize property to tune the frequency of flushing for your application's message sizes and frequency.

**C API** TipcSrvConnFlush

**Example** This example flushes the connection to an RTserver:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
...
SrvConn.Flush
...
SrvConn.Close
```

## GetBufferReadSize Method

The GetBufferReadSize method returns the total number of bytes in a connection's read buffer.

**Syntax** *object.GetBufferReadSize (read\_buffer\_return)*

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*read\_buffer\_return* is a long integer used to hold the returned value.

**Return Type** None

**Remarks** GetBufferReadSize returns the total number of bytes in a connection's read buffer.

**C API**      TipcSrvConnBufferGetReadSize

**Example**      This example returns the total number of bytes in a connection’s read buffer:

```
Dim read_buf as Long
Dim SrvConn as new TipcSvcConn
Dim Svc as new TipcSvc

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
...
SrvConn.GetBufferReadSize (read_buf)
...
SrvConn.Close
```

GetBufferWriteSize Method

The GetBufferWriteSize method returns the total number of bytes in a connection’s write buffer.

**Syntax**      *object*.GetBufferWriteSize (*write\_buffer\_return*)  
where:  
*object*                      is an object expression that evaluates to a TipcSrvConn object.  
*write\_buffer\_return* is a long integer used to hold the returned value.

**Return Type**      None

**Remarks**      GetBufferWriteSize returns 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.

**C API**      TipcSrvConnBufferGetWriteSize

**Example**      This example returns the total number of bytes in a connection’s write buffer:

```
Dim write_buf as Long
Dim SrvConn as new TipcSvcConn
Dim Svc as new TipcSvc

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
...
SrvConn.GetBufferWriteSize (write_buf)
...
SrvConn.Close
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## GetSubscribe Method

The GetSubscribe method returns whether or not an RTclient is subscribing to a subject.

**Syntax** *object.GetSubscribe subject status*

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*subject* is the subject to check whether or not this connection is subscribed to.

*status* is a Boolean status returned:

- True indicates that this TipcSrvConn is currently subscribed to the subject.
- False indicates it is not.

**Return Type** None

**C API** TipcSrvConnSubjectGetSubscribe

**Example** This example checks whether the client is subscribed to the "/subject" subject:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim status as Boolean

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
'''
SrvConn.GetSubscribe "/subject", status
If status is True then
 Debug.Print "Subscribed!"
End If
'''
SrvConn.Close
```

## GetSubscribeLb Method

The GetSubscribe method returns whether or not an RTclient is subscribing to a subject, including load balancing information.

**Syntax** *object.GetSubscribeLb subject, subject\_status, lb\_status*

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

|                       |                                                                                                                                                                                                       |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>subject</i>        | is the subject to check whether or not this connection is subscribed to.                                                                                                                              |
| <i>subject_status</i> | Is a Boolean status returned: <ul style="list-style-type: none"><li>• True indicates that this TipcSrvConn is currently subscribed to the subject.</li><li>• False indicates it is not.</li></ul>     |
| <i>lb_status</i>      | is a Boolean status returned: <ul style="list-style-type: none"><li>• True indicates that this TipcSrvConn is currently load balancing on the subject.</li><li>• False indicates it is not.</li></ul> |

**Return Type**    None

**C API**    TipcSrvConnSubjectGetSubscribe

**Example**    This example checks if the client is subscribed and load-balancing on the "/subject" subject:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim status as Boolean
Dim lb_status as Boolean

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open

...
SrvConn.GetSubscribeLb "/subject", status, lb_status
If status is True then
 Debug.Print "Subscribed!"
End If
If lb_status is True then
 Debug.Print "Load balancing!"
End If

...
SrvConn.Close
```

## GetTrafficBytesRecv Method

The GetTrafficBytesRecv method returns the total number of bytes received on a connection.

**Syntax**    *object*.GetTrafficBytesRecv (*bytes\_recv\_return*)  
where:  
*object*    is an object expression that evaluates to a TipcSrvConn object.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

*bytes\_rcv\_return* is a long integer used to hold the returned value.

|             |                                                                                                                                                                                                                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Return Type | None                                                                                                                                                                                                                                                                                                         |
| Remarks     | GetTrafficBytesRecv returns the total number of bytes received on a connection.                                                                                                                                                                                                                              |
| C API       | TipcSrvConnGetTrafficBytesRecv                                                                                                                                                                                                                                                                               |
| Example     | <p>This example returns the total number of bytes received on a connection:</p> <pre>Dim bytes_return as Long Dim SrvConn as new TipcSvcConn Dim Svc as new TipcSvc  Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "") SrvConn.Open ... SrvConn.GetTrafficBytesRecv (bytes_return) ... SrvConn.Close</pre> |

## GetTrafficBytesSent Method

The GetTrafficBytesSent method returns the total number of bytes sent on a connection.

|             |                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p><i>object</i>.GetTrafficBytesSent (<i>bytes_sent_return</i>)</p> <p>where:</p> <p><i>object</i> is an object expression that evaluates to a TipcSrvConn object.</p> <p><i>bytes_sent_return</i> is a long integer used to hold the returned value.</p> |
| Return Type | None                                                                                                                                                                                                                                                      |
| Remarks     | GetTrafficBytesSent returns the total number of bytes sent on a connection.                                                                                                                                                                               |
| C API       | TipcSrvConnGetTrafficBytesSent                                                                                                                                                                                                                            |
| Example     | <p>This example returns the total number of bytes sent on a connection:</p> <pre>Dim bytes_return as Long Dim SrvConn as new TipcSvcConn Dim Svc as new TipcSvc</pre>                                                                                     |

```
Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
...
SrvConn.GetTrafficBytesSent (bytes_return)
...
SrvConn.Close
```

## GetTrafficMsgsRecv Method

The GetTrafficMsgsRecv method returns the total number of messages received on a connection.

**Syntax**     *object*.GetTrafficMsgsRecv (*msgs\_recv\_return*)  
              where:  
              *object*                is an object expression that evaluates to a TipcSrvConn object.  
              *msgs\_recv\_return* is a long integer used to hold the returned value.

**Return Type**     None

**Remarks**     GetTrafficMsgsRecv returns the total number of messages received on a connection.

**C API**     TipcSrvConnGetTrafficMsgsRecv

**Example**     This example returns the total number of messages received on a connection:

```
Dim msgs_return as Long
Dim SrvConn as new TipcSvcConn
Dim Svc as new TipcSvc

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
...
SrvConn.GetTrafficMsgsRecv (msgs_return)
...
SrvConn.Close
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED



## GetTrafficMsgsSent Method

---

The GetTrafficMsgsSent method returns the total number of messages sent on a connection.

**Syntax**     *object*.GetTrafficMsgsSent (*msgs\_sent\_return*)  
where:  
*object*             is an object expression that evaluates to a TipcSrvConn object.  
*msgs\_sent\_return* is a long integer used to hold the returned value.

**Return Type**     None

**Remarks**     GetTrafficMsgsSent returns the total number of messages sent on a connection.

**C API**     TipcSrvConnGetTrafficMsgsSent

**Example**     This example returns the total number of messages sent on a connection:

```
Dim msgs_return as Long
Dim SrvConn as new TipcSvcConn
Dim Svc as new TipcSvc

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
...
SrvConn.GetTrafficMsgsSent (msgs_return)
...
SrvConn.Close
```

## GmdFileCreate Method

---

The GmdFileCreate method creates a GMD area for the connection to an RTserver.

**Syntax**     *object*.GmdFileCreate  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type**     None

**Remarks**     This is normally handled automatically, but is included to facilitate fine-grained GMD error control. For more information on GMD areas, see the *TIBCO SmartSockets User's Guide*.

**C API**     TipcSrvConnGmdFileCreate

**Example** This example creates the GMD file area before sending the first GMD message:

```
Dim msg As TipcMsg
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.GmdFileCreate
Set msg = Svc.MtLookup("info")
msg.DeliveryMode = tipcDeliveryAll
SrvConn.MsgSend
SrvConn.Flush
```

## GmdFileDelete Method

The GmdFileDelete method deletes a GMD area created with the GmdFileCreate method.

**Syntax** `object.GmdFileDelete`  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** None

**Remarks** For more information on the GMD area, see the *TIBCO SmartSockets User's Guide*.

**C API** TipcSrvConnGmdFileDelete

**Example** This example removes a GMD area before sending a message:

```
Dim msg As TipcMsg
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.GmdFileDelete
Set msg = Svc.MtLookup("info")
msg.DeliveryMode = tipcDeliveryAll
SrvConn.MsgSend
SrvConn.Flush
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## GmdMsgDelete Method

---

The GmdMsgDelete method deletes a guaranteed message from the GMD area after a GMD failure.

**Syntax** `object.GmdMsgDelete msg`

where:

`object` is an object expression that evaluates to a TipcSrvConn object.

`msg` is class TipcMsg and is the message to delete from the GMD area.

**Return Type** None

**Remarks** For more information on deleting a message from the GMD area, see the *TIBCO SmartSockets User's Guide*.

**C API** TipcSrvConnGmdMsgDelete

**Example** This example deletes a GMD message:

```
TipcSvc1.Srv.GmdMsgDelete msg
```

## GmdMsgServerDelete Method

---

The GmdMsgServerDelete method deletes a GMD message on the RTserver after a GMD failure.

**Syntax** `object.GmdMsgServerDelete msg`

where:

`object` is an object expression that evaluates to a TipcSrvConn object.

`msg` is class TipcMsg and is the message to delete from the RTserver's GMD area.

**Return Type** None

**Remarks** For more information on deleting a failed GMD message from RTserver, see the *TIBCO SmartSockets User's Guide*.

**C API** TipcSrvConnGmdMsgServerDelete

**Example** This example deletes a GMD message on RTserver:

```
SrvConn.GmdMsgServerDelete msg
```

## GmdMsgStatus Method

---

The GmdMsgStatus method polls RTserver for the GMD status of a message. After this call, the RTserver sends a message of type T\_MT\_GMD\_STATUS\_RESULT back to the calling RTclient, which can be processed for the names of RTclients that have acknowledged receipt of the specified message.

**Syntax** *object*.GmdMsgStatus (*msg*)  
where:  
*object* is an object expression that evaluates to a TipcSrvConn object.  
*msg* is class TipcMsg and is the message to delete from the GMD area.

**Return Type** None

**C API** TipcSrvConnGmdMsgStatus

**Example** This example creates a callback to handle the response message, gmd\_status\_result, from this method:

```
Dim mt As TipcMt
Dim svc as new TipcSvc
Set mt = svc.MtLookup "gmd_status_result"
...
SrvConn.AddProcessCb cb, mt, 1, nothing
SrvConn.GmdMsgStatus msg
```

## KeepAlive Method

---

The KeepAlive method checks to determine if the connection to an RTserver is still available.

**Syntax** *object*.KeepAlive  
where *object* is an object expression that evaluates to a TipcSrvConn object.

|                    |                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Return Type</b> | Boolean value, where:<br><b>True</b> indicates the connection to the RTserver is intact.<br><b>False</b> indicates the connection to the RTserver was lost.                                                                  |
| <b>Remarks</b>     | Upon returning <b>False</b> , the global SmartSockets error number is set and any applicable TipcError callback events are invoked.                                                                                          |
| <b>C API</b>       | TipcSrvConnKeepAlive                                                                                                                                                                                                         |
| <b>Example</b>     | This example makes sure that the connection to the RTserver is still intact:<br><pre>Dim alive As Boolean  alive = SrvConn.KeepAlive If alive = False Then     MsgBox "Connection is broken!", 0, "No RTserver" End If</pre> |

## MainLoop Method

---

The MainLoop method reads and processes the SmartSockets messages present on the connection to an RTserver. This is a convenience function that can almost always be used in place of the combination of MsgNext, MsgProcess and MsgDestroy.

|                    |                                                                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <i>object</i> .MainLoop ( <i>timeout</i> )<br>where:<br><i>object</i> is an object expression that evaluates to a TipcSrvConn object.<br><i>timeout</i> is a double real number that specifies the maximum time in seconds to wait for messages to arrive. Specify 0 to perform one round of processing and return immediately. |
| <b>Return Type</b> | None                                                                                                                                                                                                                                                                                                                            |
| <b>C API</b>       | TipcSrvConnMainLoop                                                                                                                                                                                                                                                                                                             |
| <b>Example</b>     | This example processes all queued messages:<br><pre>SrvConn.MainLoop 0.0</pre>                                                                                                                                                                                                                                                  |

## MsgInsert Method

The MsgInsert method inserts a message into the queue of messages on the connection to an RTserver, at the specified position.

**Syntax**     *object*.MsgInsert *msg*, *position*  
where:  
*object*     is an object expression that evaluates to a TipcSrvConn object.  
*msg*        is class TipcMsg and is the message to be inserted into the queue.  
*position*    is a long integer that specifies where (which position) in the queue to insert the message. Specify 0 to insert the message into the first position in the queue.

**Return Type**     None

**Remarks**        The message must not be destroyed by calling the MsgDestroy method, as MsgInsert takes responsibility for destroying the message.

**C API**            TipcSrvConnMsgInsert

**Example**        This example inserts a message at the front of the message queue:

```
Dim msg As TipcMsg
Dim svc as new TipcSvc

Set msg = svc.MsgCreate(svc.MtLookup("info"))
msg.AppendStr "Informational string"
SvcConn.MsgInsert msg, 0
```

## MsgNext Method

The MsgNext method retrieves the next message sent to an RTclient by RTserver.

**Syntax**     *object*.MsgNext (*timeout*)  
where:  
*object*     is an object expression that evaluates to a TipcSrvConn object.  
*timeout*    is a double real number that specifies the maximum time, in seconds, to wait for messages to arrive.

**Return Type**     TipcMsg

**Remarks**        Destroy the message returned by MsgNext when it is no longer needed.

**C API**     TipcSrvConnMsgNext

**Example**     This example retrieves a message, in the queue or arriving in the next two seconds:

```
Dim msg as TipcMsg
Set msg = SrvConn.MsgNext(2.0)
if NOT msg is Nothing Then
 SrvConn.MsgProcess msg
End If
```

## MsgProcess Method

The MsgProcess method processes the specified message by firing any registered callback events, including default callbacks.

**Syntax**     *object*.MsgProcess *msg*

where:

*object*     is an object expression that evaluates to a TipcSrvConn object.

*msg*        is class TipcMsg and is the message to process.

**Return Type**     None

**C API**     TipcSrvConnMsgProcess

**Example**     This example retrieves a message and processes it:

```
Dim msg as TipcMsg
Set msg = SrvConn.MsgNext(2.0)
SrvConn.MsgProcess msg 'Process message
```

## MsgSearchType Method

The MsgSearchType method returns the first (by priority) message in the message queue of the specified type or waits up to the specified amount of time for a message of the desired type to arrive, and then returns that message.

**Syntax**     *object*.MsgSearchType (*timeout*, *message\_type*)

where:

*object*        is an object expression that evaluates to a TipcSrvConn object.

*timeout* is a double real number specifying the maximum time in seconds to wait for messages of *message\_type* to arrive, if none are in the queue.

*message\_type* is class TipcMt and specifies the message type to retrieve.

**Return Type** TipcMsg

**C API** TipcSrvMsgSearchType

**Example** This example processes the first info message received within the next 10 seconds:

```
Dim mt As New TipcMt
Dim msg as TipcMsg
Dim srv as TipcSrvConn

...
mt.Lookup "info"
set msg = Srv.MsgSearchType(10.0, mt)
if not msg is nothing Then
 srv.MsgProcess msg
End If
```

### MsgSend Method

The MsgSend method sends the specified message to an RTserver. This is how RTclient publishes messages to subjects.

**Syntax** *object*.MsgSend *msg*

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*msg* is class TipcMsg and is the message to be published.

**Return Type** None

**Remarks** The value of the parent's TipcSvc's ServerMsgSend property controls whether this message is sent.

**C API** TipcSrvConnMsgSend



**Example** This example sends a quit force message to RTserver terminating all RTclients in the same project:

```
Dim msg as TipcMsg
Dim svc as new TipcSvc

Set msg = svc.MsgCreate(svc.MtLookup("control"))
msg.AppendStr "quit force"
msg.Dest = "/..."
svc.MsgSend msg
svc.Flush
```

## MsgSendRpc Method

The MsgSendRpc method performs a remote procedure call (RPC) with messages, from RTserver. The specified message is sent to the server and program operation halts until the timeout expires or the response message is received.

**Syntax** *object.MsgSendRpc (msg, timeout)*

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*msg* is class TipcMsg and is the RPC message to be sent.

*timeout* is a double real number that specifies the maximum time in seconds to wait for the RPC response message to return.

**Return Type** TipcMsg

**Remarks** The numeric message ID of the RPC response must be equal to one greater than the ID of the RPC request.

**C API** TipcSrvConnMsgSendRpc

**Example** This example sends a request message and waits to 30 seconds for a response to come back:

```
Dim mt1 As New TipcMt
Dim mt2 As New TipcMt
Dim msg1 As New TipcMsg
Dim msg2 As TipcMsg

Set mt1 = svc.MtCreate "request", 1050, ""
Set mt2 = svc.MtCreate "response", 1051, "int4"
Set msg1 = svc.MsgCreate mt1
msg1.Dest = "/requestor"
Set msg2 = SrvConn.MsgSendRpc msg1, 30.0
If Not msg2 Is Nothing Then
 SrvConn.MsgProcess msg2
End If
```

## Open Method

The Open method closes a connection to RTserver. Any Open callbacks are called at this time.

**Syntax**     *object*.Open  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type**     None

**C API**     TipcSrvConn Open

**Example**     This example creates, opens, and closes a TipcSrvConn:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
SrvConn.Close
```

## OpenCbCreate Method

The OpenCbCreate method registers a user-defined callback to be called when a connection is opened. Multiple callbacks may be registered. See TipcConnChangeCb.

**Syntax**     *object*.OpenCbCreate *event\_handler*, *priority*, *arg*  
where:  
*object*             is an object expression that evaluates to a TipcSrvConn object.  
*event\_handler*     is a user-defined callback object, implemented from the TipcConnChangeCb class.  
*priority*           is a long representing the priority of the object.  
*arg*                is a variant user argument that is passed to the ConnChange event in the user-defined callback class implemented from TipcConnChangeCb.

**Return Type**     Long (callback tag used to remove a callback if required)

**C API**     TipcSrvConnOpenCbCreate

**Example** This example creates and registers an open callback and opens, and closes a TipcSrvConn. The open callback is invoked during the SrvConn.Open method:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim cb_obj as new MyCallBackObject
Dim cb_tag as long

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
cb_tag = SrvConn.OpenCbCreate cb_obj, 1, "argument"
SrvConn.Open
SrvConn.Close
```

## ProcessCbCreate Method

The ProcessCbCreate method installs an event handler to process SmartSockets messages of a specific type. The TipcMsg event of the TipcProcessCb object specified is invoked when the RTclient receives a message of the given type. Multiple callbacks may be registered. See TipcProcessCb.

**Syntax** *object*. ProcessCbCreate *event\_handler*, *mt*, *priority*, *arg*  
 where:  
*object* is an object expression that evaluates to a TipcSrvConn object.  
*event\_handler* is a user-defined callback object, implemented from the TipcConnChangeCb class  
*mt* evaluates to a TipcMt object that specifies the type of message that this process event handles. If message\_type is null, the process event handles all message types.  
*priority* is a long representing the priority of the callback.  
*arg* is a variant user argument that is passed to the TipcMsg event in the user-defined callback class implemented from TipcProcessCb.

**Return Type** Long (callback tag used to remove a callback if required)

**C API** TipcSrvConnProcessCbCreate

**Example** This example registers a process callback. The process callback will be invoked anytime a message is received in the MainLoop method, or MsgProcess method:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim cb_obj as new MyCallBackObject
Dim cb_tag as long
Dim Mt as TipcMt
```

```
Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
Set mt = Svc.MtLookup("info")
cb_tag = SrvConn.ProcessCbCreate cb_obj, mt, 1, "argument"
SrvConn.Open
SrvConn.Close
```

## SetCredentials Method

The SetCredentials method sets the credentials of a TipcSrvConn. The credentials are passed to the Security Manager to authenticate the connection. If authentication fails, the Open method fails when called.

**Syntax**    *object.SetCredentials auth\_policy\_id, auth\_data, auth\_data\_len*  
where:  
*object*            is an object expression that evaluates to a TipcSrvConn object.  
*auth\_policy\_id*    is a 4 byte long integer that represents the policy ID for Entitlement.  
*auth\_data*        is a Variant containing Entitlement data.  
*auth\_data\_len*    is a 4 byte long integer specifying the length of the auth\_data variant, in bytes.

**Return Type**    None

**C API**            TipcSrvConnSetCredentials

**Example**        This example sets credentials in a TipcSrvConn for authentication with a server:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim auth_data as Variant

Auth_Data = "123456789"

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.SetCredentials 1, auth_data, 9
SrvConn.Open
SrvConn.Close
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## SetStdSubjectSubscribe Method

---

The SetStdSubjectSubscribe method starts or stops subscribing to standard subjects.

**Syntax** *object*.SetStdSubjectSubscribe *status*

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*status* is a Boolean value that indicates whether to start or stop subscribing to standard subjects.

**Return Type** None

**C API** TipcSrvConnStdSubjectSetSubscribe

**Example** This example subscribes a TipcSrvConn to standard subjects:

```
Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
SrvConn.SetStdSubjectSubscribe True
SrvConn.Close
```

## SetUsernamePassword Method

---

The SetUsernamePassword method sets the username and password used by RTserver to authenticate the RTclient connection.

**Syntax** *object*.SetUsernamePassword *username*, *password*

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*username* is the username to use when making a connection to RTserver.

*password* is the password for use with *username*.

**Return Type** None

**C API** TipcSrvConnSetUsernamePassword

**Example** This example sets the username and password for the RTserver connection, then opens the connection:

```
Dim RTserver_Conn As TipcSrvConn
Dim Svc as new TipcSvc

Set RTserver_Conn = Svc.TipcSrvConnCreate("", "", "", "")

RTserver_Conn.SetUsernamePassword ("admin"), ("smartsockets")

Rtserver_Conn.Open
```

## SrvIsRunning Method

---

The SrvIsRunning method checks to see whether an RTClient can connect to RTserver.

**Syntax** *object*.SrvIsRunning  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** Boolean - whether or not the server is running

**C API** TipcSrvIsRunning

**Example** This example checks to see if an RTserver is running:

```
Dim SrvConn as TipcSrvConn
Dim svc as New TipcSvc
Dim status as Boolean

Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
Status = SrvConn.SrvIsRunning
If Status = True then
 Debug.Print "Server Is Running"
End If
```

## SrvLock Method

---

The SrvLock method acquires exclusive access to a connection.

**Syntax** *object*.SrvLock  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

**Remarks** TipcSrvConnLock must be balanced with a TipcSrvConnUnlock.

**C API** TipcSrvConnLock

**Example** This example locks a TipcSrvConn:

```
Dim SrvConn as TipcSrvConn
Dim svc as New TipcSvc
Dim status as Boolean

Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
SrvConn.SrvLock
```

## TipcSrvMonCreate Method

The TipcSrvMonCreate method creates a new TipcSrvMon object associated with the TipcSrvConn object. The new object is used for monitoring calls on the TipcSrv object.

**Syntax** *object*.TipcSrvMonCreate

where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** None

**C API** There is no equivalent C API for the TipcSrvMonCreate method.

**Example** See the example for ClientExtPoll Method on page 473.

## TipcSrvMonExtCreate Method

The TipcSrvMonExtCreate method creates a new TipcSrvMonExt object associated with the TipcSrvConn object. The new object is used for defining or setting RTclient extension data on the TipcSrvConn object.

**Syntax** *object*.TipcSrvMonExtCreate

where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** None

**C API** There is no equivalent C API for the TipcSrvMonExtCreate method.

**Example** See the example for ClientExtPoll Method on page 473.

## SrvUnlock Method

---

The SrvUnlock method releases exclusive access to a connection.

**Syntax** `object.SrvUnlock`  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** None

**C API** TipcSrvConnUnlock

**Example** This example unlocks a TipcSrvConn:

```
Dim SrvConn as TipcSrvConn
Dim svc as New TipcSvc
Dim status as Boolean

Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
SrvConn.SrvLock
SrvConn.SrvUnlock
```

## SubjectCbCreate Method

---

The SubjectCbCreate method installs an event handler to process SmartSockets messages sent to a specified subject with a specified message type. The TipcMsg event of the TipcProcessCb object specified is invoked when a message of the given subject and type is received. Multiple callbacks may be registered. See TipcProcessCb.

**Syntax** `object.SubjectCbCreate event_handler, subject, mt, priority, arg`  
where:  
*object* is an object expression that evaluates to a TipcSrvConn object.  
*event\_handler* is a user-defined callback object, implemented from the TipcConnChangeCb class.  
*subject* evaluates to a TipcMt object that specifies the type of message that this process event handles. If *message\_type* is null, the process event handles all message types.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED



*mt* is a string that specifies the subject for which to create a callback. If subject is null, callbacks are created for all subjects.

*priority* is a long representing the priority of the object.

*arg* is a variant user argument that is passed to the TipcMsg event in the user-defined callback class implemented from TipcProcessCb.

**Return Type** Long (callback tag used to remove a callback if required)

**C API** TipcSrvConnSubjectCbCreate

**Example** This example creates callbacks for all info messages, all messages, and messages with the subject "/stocks":

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim cb_obj as new MySubjectCallback1
Dim cb_obj2 as new MySubjectCallback2
Dim cb_obj3 as new MySubjectCallback3
Dim cb_tag as long
Dim cb_tag2 as long
Dim cb_tag3 as long

Dim Mt as TipcMt

SrvConn.Open

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
Set mt = Svc.MtLookup("info")
' create a subject callback for all info messages received on /stocks
cb_tag = SrvConn.SubjectCbCreate cb_obj, "/stocks", mt, 1, "argument"
' create a subject callback for all info messages received
cb_tag2 = SrvConn.SubjectCbCreate cb_obj2, "/...", mt, 1, "argument"
' create a subject callback for all messages received
cb_tag3 = SrvConn.SubjectCbCreate cb_obj3, Nothing, Nothing, 1, Null

SrvConn.Close
```

## SubjectCbDestroyAll Method

The SubjectCbDestroyAll method uninstalls all subject callbacks that were previously installed with AddSubjectCb or AddSubjectDefaultCb.

**Syntax** *object*.SubjectCbDestroyAll  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** None

**C API** None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

**Example** This example destroys all previously registered subject callbacks:

```
Dim s as TipcSrvConn
...
s. SubjectCbDestroyAll
```

## SubjectDefaultCbCreate Method

The SubjectDefaultCbCreate method installs an event handler to process SmartSockets messages that have no corresponding subject callback. The TipcMsg event of the TipcProcessCb object specified is invoked when a message not processed by any other subject callback is received.

**Syntax** *object*.SubjectCbCreate *event\_handler*, *priority*, *arg*  
where:  
*object* is an object expression that evaluates to a TipcSrvConn object.  
*event\_handler* is a user-defined callback object, implemented from the TipcConnChangeCb class.  
*priority* is a Long representing the priority of the object.  
*arg* is a variant user argument that is passed to the TipcMsg event in the user-defined callback class implemented from TipcProcessCb.

**Return Type** Long (callback tag used to remove a callback if required)

**C API** TipcSrvConnSubjectDefaultCbCreate

**Example** This example creates callbacks for all info messages, all messages, and messages with the subject "/stocks":

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim cb_obj as new MySubjectCallback1
Dim cb_tag as long

Dim Mt as TipcMt

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
Set mt = Svc.MtLookup("info")
SrvConn.Open
cb_tag = SrvConn.SubjectDefaultCbCreate cb_obj, 1, "argument"
SrvConn.Close
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## SubjectGmdInit Method

---

The SubjectGmdInit method initializes GMD accounting for the specified subject.

**Syntax**     `object.SubjectGmdInit subject`

where:

*object*     is an object expression that evaluates to a TipcSrvConn object.

*subject*    is a string that specifies the subject for initializing GMD.

**Return Type**     None

**Remarks**        This method is not usually used by application developers because publishing with GMD automatically starts GMD accounting. It can, however, be used to prime the RTserver to be ready for publishing large numbers of GMD messages. For more information on GMD, see the *TIBCO SmartSockets User's Guide*.

**C API**            TipcSrvConnSubjectGmdInit

**Example**         This example initializes GMD for the "/stocks" subject:

```
Dim s As TipcSrvConn
...
s.SubjectGmdInit "/stocks"
```

## SubjectLbInit Method

---

The SubjectLbInit method initializes load balancing accounting for the specified subject.

**Syntax**     `object.SubjectLbInit subject`

where:

*object*     is an object expression that evaluates to a TipcSrvConn object.

*subject*    is a string that specifies the subject for initializing load balancing accounting.

**Return Type**     None

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Remarks | This method is not usually used by application developers because publishing with load balancing automatically starts load balancing accounting. It can, however, be used to prime the RTserver to be ready for publishing large numbers of load balanced messages. Currently, both GMD and load balancing accounting use the same mechanism—choose the appropriate method for the desired action. For more information on load balancing, see the <i>TIBCO SmartSockets User's Guide</i> . |
| C API   | TipcSrvConnSubjectLbInit                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Example | This example initializes load balancing for the <code>"/stocks"</code> subject:<br><pre>Dim s As TipcSrvConn ... s.SubjectLbInit "/stocks"</pre>                                                                                                                                                                                                                                                                                                                                            |

## SubjectSetSubscribe Method

The SubjectSetSubscribe method subscribes a connection to a subject.

|        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax | <pre>object.SubjectSetSubscribe subject, status</pre> <p>where:</p> <p><i>object</i> is an object expression that evaluates to a TipcSrvConn object.</p> <p><i>subject</i> is a string representing the subject to subscribe to.</p> <p><i>status</i> is a Boolean status to determine whether or not to subscribe to the subject:</p> <ul style="list-style-type: none"><li>• True causes this TipcSrvConn to subscribe to the subject.</li><li>• False stops this TipcSrvConn from subscribing to the subject.</li></ul> |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|             |                                |
|-------------|--------------------------------|
| Return Type | None                           |
| C API       | TipcSrvConnSubjectSetSubscribe |

|         |                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Example | This example subscribes and unsubscribes a TipcSrvConn to the <code>"/stocks"</code> subject:<br><pre>Dim SrvConn as TipcSrvConn Dim Svc as new TipcSvc Dim status as Boolean  Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "") SrvConn.Open  ' start subscribing to /stocks SrvConn.SubjectSetSubscribe "/stocks", True ' stop subscribing to /stocks SrvConn.SubjectSetSubscribe "/stocks", False ...</pre> |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

SrvConn.Close

## SubjectSetSubscribeEx Method

The SubjectSetSubscribeEx method starts or stops an RTclient from subscribing to a subject using extended options.

|                       |                                                                                                                                                                                                                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>         | <i>object.SubscribeSetSubscribeEx subject, status, subscribe_type, flags, type_specific_struct</i>                                                                                                                                                                                                                                      |
|                       | where:                                                                                                                                                                                                                                                                                                                                  |
| <i>object</i>         | is an object expression that evaluates to a TipcSrvConn object.                                                                                                                                                                                                                                                                         |
| <i>subject</i>        | is a string representing the subject name to set the subscribe status of.                                                                                                                                                                                                                                                               |
| <i>status</i>         | is a Boolean value set to determine whether or not to subscribe to the subject: <ul style="list-style-type: none"><li>• True causes this TipcSrvConn to subscribe to the subject.</li><li>• False stops this TipcSrvConn from subscribing to the subject.</li></ul>                                                                     |
| <i>subscribe_type</i> | is the type of subscription: <ul style="list-style-type: none"><li>• tipcSubjectNone performs a normal subscribe</li><li>• tipcSubjectCache performs a subscribe using SmartSockets Cache.</li></ul>                                                                                                                                    |
| <i>flags</i>          | is a Long value used to set flags for the cache: <ul style="list-style-type: none"><li>• 0 sets the flag to 0 when <i>subscribe_type</i> is tipcSubjectNone</li><li>• 2 for force initial load</li><li>• 3 retrieves the last message from the cache or a new message if one arrives before the cache retrieval has completed</li></ul> |
|                       | <i>type_specific_struct</i> is a Variant, reserved.                                                                                                                                                                                                                                                                                     |
|                       | <b>Note:</b> 1 is currently not a valid flag.                                                                                                                                                                                                                                                                                           |
| <b>Return Type</b>    | None                                                                                                                                                                                                                                                                                                                                    |
| <b>C API</b>          | TipcSrvConnSubjectSetSubscribeEx                                                                                                                                                                                                                                                                                                        |

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

**Example** This example subscribes and unsubscribes a TipcSrvConn to the `"/stocks"` subject, and requests caching on the `"/stocks"` subject from a SmartSockets Cache server:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim status as Boolean

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
...
SrvConn.SubjectSetSubscribeEx "/stocks", True, tipcSubjectCache, 0,
Nothing
...
SrvConn.Close
```

## SubjectSetSubscribeLb Method

The SubjectSetSubscribeLb method subscribes a connection to a subject, and determines whether or not to load balance the subject.

**Syntax** `object.SubjectSetSubscribeLb subject, status, lb_status`  
where:  
`object` is an object expression that evaluates to a TipcSrvConn object.  
`subject` is a string representing the subject to subscribe to.  
`status` is a Boolean value set to determine whether or not to subscribe to the subject:

- `True` causes this TipcSrvConn to subscribe to the subject.
- `False` stops this TipcSrvConn from subscribing to this subject.

`lb_status` is a Boolean value set to determine whether or not to load balance the subject. `True` causes this subject to be load balanced.

**Return Type** None

**C API** TipcSrvConnSubjectSetSubscribeLb

**Example** This example subscribes to the `"/stocks"` subject with load balancing, then disables load balancing on the `"/stocks"` subject, and finally unsubscribes the TipcSrvConn to the `"/stocks"` subject:

```
Dim SrvConn as TipcSrvConn
Dim Svc as new TipcSvc
Dim status as Boolean

Set SrvConn = Svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
```

```
'''
' start subscribing to /stocks load balanced
SrvConn.SubjectSetSubscribeLb "/stocks", True, True
' start turn off load balancing on the /stocks subject
SrvConn.SubjectSetSubscribeLb "/stocks", True, False
' stop subscribing to /stocks
SrvConn.SubjectSetSubscribeLb "/stocks", False, False
'''
SrvConn.Close
```

## TipcSrvConn Class Properties

---

### Arch Property

---

The Arch property contains a string indicating the hardware architecture on which RTserver is running and is used to handle conversion to and from machines with differing data formats. This property is read-only.

**Syntax**     *object*.Arch  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Data Type**     String

**C API**     TipcSrvConnGetArch

**Example**     This example displays the architecture of RTserver:  
`Debug.Print "RTserver runs " + SrvConn.Arch`

### AutoFlushSize Property

---

The AutoFlushSize property retrieves or sets the number of outgoing bytes buffered before being automatically flushed to an RTserver.

**Syntax**     *object*.AutoFlushSize [=flush\_size]  
where:  
*object*     is an object expression that evaluates to a TipcSrvConn object.  
*flush\_size* is an integer that specifies the amount of outgoing data, in bytes, to buffer before automatically flushing the connection to the RTserver. The default value is 8192.

**Data Type**     Long (Integer)

**C API**     TipcSrvConnGetAutoFlushSize, TipcSrvConnSetAutoFlushSize



**Example** This example doubles the buffer size before autoflushing, until it is at least 16K:

```
While (SrvConn.AutoFlushSize < 16384)_
 SrvConn.AutoFlushSize = SrvConn.AutoFlushSize * 2
Wend
```

## BlockMode Property

The BlockMode property retrieves or sets the input/output (I/O) mode used for the connection to an RTserver. The mode is blocking or non-blocking. The default value is True.

**Syntax** *object*.BlockMode [=*block\_mode*]

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*block\_mode* is a Boolean value. True sets the blocking mode and False sets the non-blocking I/O.

**Data Type** Boolean

**C API** TipcSrvConnGetBlockMode, TipcSrvConnSetBlockMode

**Example** This example sets the connection to the RTserver for non-blocking mode:

```
Dim s As TipcSrvConn
...
SrvConn.BlockMode = False
```

## DefaultProtocols Property

The DefaultProtocols property returns the default protocols for the connection.

**Syntax** *object*.DefaultProtocols [=*default\_protocols*]

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*default\_protocols* is a string representing the default protocols for the connection.

**Data Type** String

**C API** TutOptionSetStr("default\_protocols")

**Example** This example sets the DefaultProtocols of a TipcSrvConn to "tcp":

```
Dim SrvConn as TipcSrvConn
SrvConn.DefaultProtocols = "tcp"
```

## DefaultSubjectPrefix Property

The DefaultSubjectPrefix specifies the default text prepended to subject names used by RTclient that do not begin with a slash (/).

**Syntax** `object.DefaultSubjectPrefix [=prefix]`  
where:  
`object` is an object expression that evaluates to a TipcSrvConn object.  
`prefix` is a string that specifies the text to be prefixed to subjects that are not fully qualified, that is, that do not begin with a slash (/).

**Data Type** String

**Remarks** See the *TIBCO SmartSockets User's Guide* for detailed information about subject prefixes.

**C API** Default\_Subject\_Prefix option

**Example** This example adds "/StockTicker" to the beginning of each subject name that is not fully-qualified:  
`SrvConn.DefaultSubjectPrefix = "/StockTicker"`

## GmdMaxSize Property

The GmdMaxSize property retrieves or sets the maximum amount of disk space or memory the GMD area is allowed to consume. The default value is 0, meaning that there is no limit to the amount of space used. This property cannot be changed while there is an active connection to an RTserver.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

**Syntax** `object.GmdMaxSize [=max_size]`  
 where:  
*object* is an object expression that evaluates to a TipcSrvConn object.  
*max\_size* is an integer that specifies the maximum amount of memory or space, in bytes, that the GMD area is allowed to use. Using 0 sets no limit.

**Data Type** Long (Integer)

**C API** TipcSrvConnGetGmdMaxSize, TipcSrvConnSetGmdMaxSize

**Example** This example retrieves the maximum size of the GMD area:

```
Debug.Print "Max GMD size: " &
 Format(TipcSvc1.Srv.GmdMaxSize)
```

## GmdNumPending Property

---

The GmdNumPending property retrieves the number of GMD messages still pending to be sent to an RTserver. This property is read-only.

**Syntax** `object.GmdNumPending`  
 where *object* is an object expression that evaluates to a TipcSrvConn object.

**Data Type** Long (Integer)

**C API** TipcSrvConnGetGmdNumPending

**Example** This example retrieves the number of GMD messages pending:

```
Debug.Print "Number of pending GMD messages: " &
 Format(TipcSvc1.Srv.GmdNumPending)
```

## MonitorScope Property

---

The MonitorScope property specifies the level of interest for SmartSockets monitoring in monitoring categories with no parameters (for example, RTclient names poll) or a parameter of @ (for example, RTclient time watch). MonitorScope acts as a filter, allowing you to specify a subset of the subject namespace. This is useful for RTclients monitoring a subset of a large project.

**Syntax**     `object.MonitorScope [=monitor_scope]`  
where:  
`object`        is an object expression that evaluates to a `TipcSrvConn` object.  
`monitor_scope` is a string that specifies the monitoring scope of the `RTclient`.

**Data Type**     `String`

**Remarks**     For more information on monitoring scope, see the *TIBCO SmartSockets User's Guide*.

**C API**         `Monitor_Scope` option

**Example**       This example monitors only the `cars` subjects in the subject namespace:  
`SrvConn.MonitorScope = "/stocks/cars/..."`

Node Property

---

The `Node` property retrieves the node name of the connected `RTserver`, and is read-only.

**Syntax**        `object.Node`  
where `object` is an object expression that evaluates to a `TipcSrvConn` object.

**Data Type**     `String`

**C API**         `TipcSrvConnGetNode`

**Example**       This example retrieves the node name of the connected `RTserver`:

```
Dim status As TipcSrvConnStatus
Dim s As TipcSrvConn
...
status = s.ConnStatus
If status <> tipcSrvConnNone Then
 Debug.Print "Node name: " & s.node
End If
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## NumQueued Property

---

The NumQueued property retrieves the number of messages, still pending, to be read by RTclient. This property is read-only.

**Syntax** `object.NumQueued`

where *object* is an object expression that evaluates to a TipcSrvConn object.

**Data Type** Long (Integer)

**C API** TipcSrvConnGetNumPending

**Example** This example retrieves the number of messages queued:

```
Dim s As TipcSrvConn
Dim msg as TipcMsg
Set s = TipcSvc1.Srv
If s.NumQueued > 0 Then
 set msg = s.MsgNext(0.0)
 s.MsgProcess msg
 msg.Destroy
End If
```

## Pid Property

---

The Pid property retrieves the operating system Process Identification Number (PID) of the connected RTserver for diagnostic purposes. This property is read-only.

**Syntax** `object.Pid`

where *object* is an object expression that evaluates to a TipcSrvConn object.

**Data Type** Long (Integer)

**C API** TipcSrvConnGetPid

**Example** This example retrieves the PID of the connected RTserver:

```
Dim status As TipcSrvConnStatus
Dim s As TipcSrvConn
Set s = TipcSvc1.Srv
status = s.ConnStatus
If status <> tipcSrvConnNone Then
 Debug.Print "RTserver pid: " & s.pid
End If
```

## Project Property

The Project property specifies to which project the RTclient connects.

**Syntax**     *object*.Project [=*project*]  
where:  
*object*     is an object expression that evaluates to a TipcSrvConn object.  
*project*     is a string that specifies the name of the project to which RTclient connects.  
The default value is `rtworks`. This value cannot be changed while a connection to the RTserver is active.

**Data Type**     String

**C API**     TipcSrvConnSetProject, TipcSrvConnGetProject

**Example**     This example sets the project of a TipcSrvConn to "accounting":  
  

```
Dim Svc as new TipcSvc
Dim SrvConn as TipcSrvConn

Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Project = "accounting"
```

## ServerAutoConnect Property

The ServerAutoConnect property specifies whether or not the ActiveX controls should automatically try to create a connection to an RTserver.

**Syntax**     *object*.ServerAutoConnect [=*auto\_connect*]  
where:  
*object*     is an object expression that evaluates to a TipcSrvConn object.  
*auto\_connect* is a Boolean value where:

- `True` allows the client to automatically connect to the RTserver. This is the default value.
- `False` does not allow the client components to automatically connect to RTserver.

**Data Type**     Boolean

**C API**     Server\_Auto\_Connect option

**Example** This example disables the ability to automatically connect to the RTserver:

```
TipcSvc1.ServerAutoConnect = False
```

## ServerAutoFlushSize Property

The ServerAutoFlushSize property specifies the number of bytes buffered before data is automatically sent to an RTserver.

**Syntax** `object.ServerAutoFlushSize [=flush_size]`  
where:  
*object* is an object expression that evaluates to a TipcSrvConn object.  
*flush\_size* is a long integer that specifies the number of bytes to buffer before flushing data to an RTserver.

**Data Type** Long (Integer)

**C API** Server\_Auto\_Flush\_Size option

**Example** This example prints out the value of this property:

```
Debug.Print "AutoFlushSize value is: " &
Format(TipcSvc1.ServerAutoFlushSize)
```

## ServerDeliveryTimeout Property

The ServerDeliveryTimeout property specifies the default maximum interval, in seconds, allowed for all receiving processes to acknowledge a message sent with GMD.

**Syntax** `object.ServerDeliveryTimeout[=timeout]`  
where:  
*object* is an object expression that evaluates to a TipcSrvConn object.  
*timeout* is a double real number that specifies the maximum GMD acknowledgement interval in seconds. The default value is 30.0.

**Data Type** Double

**C API** Server\_Delivery\_Timeout option

**Example** This example sets the server delivery timeout to 20 seconds:

```
TipcSvc1.ServerDeliveryTimeout = 20
If TipcSvc1.ServerDeliveryTimeout <> 20 Then
 Debug.Print "Unable to assign ServerDeliveryTimeout"
End If
```

## ServerKeepAliveTimeout Property

The ServerKeepAliveTimeout property specifies the maximum amount of time to wait, in seconds, when checking to see if the connection to an RTserver is still alive.

**Syntax** *object*.ServerKeepAliveTimeout [=*timeout*]  
where:  
*object* is an object expression that evaluates to a TipcSvc object.  
*timeout* is a double real number that specifies the maximum time in seconds to wait to determine if the RTserver is still alive. The default value is 5.0.

**Data Type** Double

**C API** Server\_Keep\_Alive\_Timeout option

**Example** This example sets the server keep alive timeout to 20 seconds:

```
TipcSvc1.ServerKeepAliveTimeout = 20
If TipcSvc1.ServerKeepAliveTimeout <> 20 Then
 Debug.Print "Unable to assign ServerKeepAliveTimeout"
End If
```

## ServerMaxReconnectDelay Property

The ServerMaxReconnectDelay property specifies the maximum interval, in seconds, allowed for waiting to reconnect to RTserver in the default error callback.

**Syntax** *object*.ServerMaxReconnectDelay[=*timeout*]  
where:  
*object* is an object expression that evaluates to a TipcSrvConn object.  
*timeout* is a double real number that specifies the maximum reconnect delay interval in seconds.



|                  |                                                                                                                                                                                                                          |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Data Type</b> | Double                                                                                                                                                                                                                   |
| <b>C API</b>     | Server_Max_Reconnect_Delay option                                                                                                                                                                                        |
| <b>Example</b>   | This example sets the server max reconnect delay to 20 seconds:<br><pre>Dim SrvConn as TipcSrvConn Dim svc as New TipcSvc Set SrvConn = svc.TipcSrvConnCreateNamed("srvconn") SrvConn.ServerMaxReconnectDelay = 20</pre> |

## ServerMsgSend Property

The ServerMsgSend property specifies whether or not RTclient is allowed to send messages to an RTserver, or if messages are discarded without transmission. This is useful for backup processes that receive, but do not send messages.

|               |                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b> | <i>object</i> .ServerMsgSend [=enable_send]<br>where:<br><i>object</i> is an object expression that evaluates to a TipcSrvConn object.<br><i>enable_send</i> is a Boolean value: <ul style="list-style-type: none"> <li>• True allows the RTclient to transmit messages to the RTserver. This is the default value.</li> <li>• False does not allow the RTclient to send messages to the RTserver.</li> </ul> |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                  |                                                                                                                                                                                                                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Data Type</b> | Boolean                                                                                                                                                                                                                                                                                            |
| <b>C API</b>     | Server_Msg_Send option                                                                                                                                                                                                                                                                             |
| <b>Example</b>   | This example will not transmit any message with the MsgSendEx property:<br><pre>TipcSvc1.ServerMsgSend = False TipcSvc1.Srv.MsgSend msg 'Msg will not be sent TipcSvc1.Srv.MsgSendEx msg, True 'Msg will not be sent TipcSvc1.Srv.MsgSendEx msg, False ' Msg will be sent TipcSvc1.Srv.Flush</pre> |

## ServerNames Property

The ServerNames property specifies a list of logical connection names used to find and start RTserver.

**Syntax**     `object.ServerNames [=server_names]`  
where:  
`object`        is an object expression that evaluates to a TipcSrvConn object.  
`server_names` is a string that specifies the logical connection names used to find and start an RTserver. The default value is `_node` which specifies the local machine.

**Data Type**     String

**Remarks**     Logical connection names have the form *protocol:node:address*, which can be shortened to *protocol:node*, *protocol*, or *node*. For more information, see the *TIBCO SmartSockets User's Guide*.

**C API**         Server\_Names option

**Example**       This example connects to a remote RTserver before failing over to the local RTserver:  
`TipcSvc1.ServerNames = "tcp:workstation1, _node"`

## ServerReadTimeout Property

The ServerReadTimeout property specifies how often, in seconds, data is expected to be available for reading on the connection to an RTserver. This setting is used to check for possible network failures.

**Syntax**     `object.ServerReadTimeout [=timeout]`  
where:  
`object`        is an object expression that evaluates to a TipcSrvConn object.  
`timeout`       is a double real number that specifies how often data is expected to be available for reading, in seconds. The default value is 30.0.

**Data Type**     Double

**C API**         Server\_Read\_Timeout option

**Example** This example sets the server read timeout to 20 seconds:

```
TipcSvc1.ServerReadTimeout = 20
If TipcSvc1.ServerReadTimeout <> 20 Then
 Debug.Print "Unable to assign ServerReadTimeout"
End If
```

## ServerStartDelay Property

The ServerStartDelay property specifies the length of a delay, in seconds, between trying the different entries in the ServerNames property (when attempting to find or start an RTserver).

**Syntax** *object*.ServerStartDelay [=server\_start\_delay]

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*server\_start\_delay* is a double real number that specifies the delay, in seconds, between trying the values in the ServerNames property. The default value is 1.0.

**Data Type** Double

**C API** Server\_Start\_Delay option

**Example** This example sets the delay to 10 seconds, if not already set:

```
If TipcSvc1.ServerStartDelay <> 10 Then
 TipcSvc1.ServerStartDelay = 10
End If
```

## ServerStartMaxTimeout Property

The ServerStartMaxTimeout property specifies the maximum time, in seconds, for the ActiveX components to wait for an RTserver to finish initializing.

**Syntax** *object*.ServerStartMaxTimeout [=timeout]

where:

*object* is an object expression that evaluates to a TipcSrvConn object.

*timeout* is a double real number that specifies the delay in seconds between trying the values in the ServerNames property. The default value is 30.0.

|                  |                                                                                                                                                                                                                                    |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Data Type</b> | Double                                                                                                                                                                                                                             |
| <b>C API</b>     | Server_Start_Timeout option                                                                                                                                                                                                        |
| <b>Example</b>   | <p>This example sets the server start timeout to 20 seconds:</p> <pre>TipcSvc1.ServerStartMaxTimeout = 20 If TipcSvc1.ServerStartMaxTimeout &lt;&gt; 20 Then     Debug.Print "Unable to assign ServerStartMaxTimeout" End If</pre> |

## ServerStartMaxTries Property

The ServerStartMaxTries property specifies how many times to traverse each connection, as specified by the entries in the ServerNames property.

|               |                                                                                                                                                                                                                                                                                                                                  |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b> | <pre>object.ServerStartMaxTries [=start_max_tries]</pre> <p>where:</p> <p><i>object</i> is an object expression that evaluates to a TipcSrvConn object.</p> <p><i>start_max_tries</i> is an integer specifying the number of times to try connecting to an RTserver specified in the ServerNames property. The default is 1.</p> |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                  |                               |
|------------------|-------------------------------|
| <b>Data Type</b> | Integer                       |
| <b>C API</b>     | Server_Start_Max_Tries option |

|                |                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------|
| <b>Example</b> | <p>This example allows for three traversals of the ServerNames property list:</p> <pre>TipcSvc1.ServertStartMaxTries = 3</pre> |
|----------------|--------------------------------------------------------------------------------------------------------------------------------|

## ServerWriteTimeout Property

The ServerWriteTimeout property specifies how long, in seconds, to block the ActiveX controls from writing data to an RTserver (before initiating failure recovery). If this timeout elapses between reads, a keep-alive is sent to the RTserver.

|                  |                                                                                                                                                                                                                                                                                                                                |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>    | <p><code>object.ServerWriteTimeout [=timeout]</code></p> <p>where:</p> <p><i>object</i> is an object expression that evaluates to a <code>TipcSrvConn</code> object.</p> <p><i>timeout</i> is a double real number that specifies how long to block, in seconds, while writing to the RTserver. The default value is 30.0.</p> |
| <b>Data Type</b> | Double                                                                                                                                                                                                                                                                                                                         |
| <b>C API</b>     | Server_Write_Timeout option                                                                                                                                                                                                                                                                                                    |
| <b>Example</b>   | <p>This example sets the server write timeout to 20 seconds:</p> <pre> TipcSvc1.ServerWriteTimeout = 20 If TipcSvc1.ServerWriteTimeout &lt;&gt; 20 Then     Debug.Print "Unable to assign ServerWriteTimeout" End If </pre>                                                                                                    |

## Status Property

The Status property retrieves or sets the connection status to an RTserver.

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>    | <p><code>object.Status [=conn_status]</code></p> <p>where:</p> <p><i>object</i> is an object expression that evaluates to a <code>TipcSrvConn</code> object.</p> <p><i>conn_status</i> is class <code>TipcSrvConnStatus</code> and defines the connection to the RTserver:</p> <ul style="list-style-type: none"> <li>• 0 is <code>tipcSrvConnNone</code>, which is no connection to RTserver</li> <li>• 1 is <code>tipcSrvConnWarm</code>, which is a warm connection to RTserver</li> <li>• 2 is <code>tipcSrvConnFull</code>, which is a full connection to RTserver</li> </ul> |
| <b>Data Type</b> | <code>TipcSrvConnStatus</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Remarks</b>   | Setting this property directly allows your application to go from a full to a warm RTserver connection. To connect or fully disconnect from RTserver, it is often easier and more explicit to directly call the <code>Open</code> and <code>Close</code> methods. For more information, see the <i>TIBCO SmartSockets User's Guide</i> .                                                                                                                                                                                                                                           |
| <b>C API</b>     | <code>TipcSrvConnConnGetConnStatus</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

**Example** This example retrieves the status of the RTserver connection:

```
Dim status As TipcSrvConnStatus
Dim SrvConn as TipcSrvConn
Dim svc as new TipcSvc
Set conn = svc.TipcSrvConnCreate("", "", "", "")

status = SrvConn.Status
Select Case status
Case tipcSrvConnNone
 Debug.Print "Server connection status: None"
Case tipcSrvConnWarm
 Debug.Print "Server connection status: Warm"
Case tipcSrvConnFull
 Debug.Print "Server connection status: Full"
End Select
```

## Subjects Property

The Subjects property retrieves the subjects to which RTclient is subscribed. This property is read-only.

**Syntax** `object.Subjects`  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Data Type** TipcSubjects

**Remarks** See TipcSubjects Class Methods on page 534 for more information.

**C API** TipcSrvConnSubjectGetSubscribe, TipcSrvMonSubjectSubscribeSetWatch

**Example** This example lists the subjects to which RTclient is subscribed:

```
Dim sb as TipcSubjects

Set sb = TipcSvc1.Srv.Subjects
For t% = 0 to sb.Count - 1
 Debug.Print sb(t%)
Next t%
```

## Timeout Property

The Timeout property retrieves or sets the Read, Write, GMD Delivery or KeepAlive timeout values.

**Syntax** `object.Timeout (timeout_type) [=timeout]`  
where:  
*object* is an object expression that evaluates to a TipcSrvConn object.  
*timeout\_type* is class TipcTimeout and specifies which type of timeout to set or return:

- 0 is `tipcTimeoutRead` which sets the server read timeout
- 1 is `tipcTimeoutWrite` which sets the server write timeout
- 2 is `tipcTimeoutKeepAlive` which sets the server keep-alive timeout
- 3 is `tipcTimeoutDelivery` which sets the GMD message delivery timeout

*timeout* is a double real number which specifies the length of the timeout, in seconds.

**Data Type** Double

**Remarks** These values are also available from the parent TipcSvc object's ServerReadTimeout Property, ServerWriteTimeout Property, ServerKeepAliveTimeout Property, and ServerDeliveryTimeout Property. See the sections for those properties for further information.

**C API** TipcSrvConnGetTimeout, TipcSrvConnSetTimeout

**Example** This example sets all the timeout values to one minute:

```
Dim s As TipcSrvConn

Set s = TipcSvc1.Srv
s.Timeout tipcTimeoutRead = 60.0
s.Timeout tipcTimeoutWrite = 60.0
s.Timeout tipcTimeoutKeepAlive = 60.0
s.Timeout tipcTimeoutDelivery = 60.0

Debug.Print "Read: " & Format(s.Timeout(tipcTimeoutRead))
Debug.Print "Write: " & Format(s.Timeout (tipcTimeoutWrite))
Debug.Print "KA: " & Format(s.Timeout(tipcTimeoutKeepAlive))
Debug.Print "Del: " & Format(s.Timeout
(tipcTimeoutDelivery))
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## UniqueSubject Property

The UniqueSubject property retrieves the unique subject name of the RTclient. Direct access to this property is read-only, but the value can be modified through a SmartSockets setopt command.

**Syntax**     *object*.UniqueSubject  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Data Type**     String

**Remarks**     See ParseStr Method on page 545 and UniqueSubject Property on page 452 for more information on setting the UniqueSubject property.

**C API**     TipcSrvConnGetUniqueSubject

**Example**     This example retrieves the unique subject for RTclient:  

```
Debug.Print "Connected to the RTserver with subject: " &
 Format(TipcSvc1.Srv.UniqueSubject)
```

## User Property

The User property retrieves the user name of the connected RTserver and is useful for diagnostic purposes. This property is read-only.

**Syntax**     *object*.User  
where *object* is an object expression that evaluates to a TipcSrvConn object.

**Data Type**     String

**C API**     TipcSrvConnGetUser

**Example**     This example retrieves the user name of the connected RTserver:  

```
Dim status As TipcSrvConnStatus
Dim s As TipcSrvConn

Set s = TipcSvc1.Srv
status = s.ConnStatus
If status <> tipcSrvConnNone Then
 Debug.Print "User: " & s.User
End If
```



## XtSource Property

---

The XtSource property returns a source suitable for XtAppAddInput from the connection. This is a read-only property.

**Syntax**     *object*.XtSource

where *object* is an object expression that evaluates to a TipcSrvConn object.

**Data Type**     Long (Integer)

**C API**     TipcSrvConnGetXtSource

**Example**     This example assigns the XtSource property of a TipcSrvConn to a Long:

```
Dim xt_source as Long
Dim SrvConn as TipcSrvConn
...
Xt_source = SrvConn.XtSource
```

## TipcSrvConnCache Class

TipcSrvConnCache — class for caching data into the SmartSockets Cache process.

**Description** The TipcSrvConnCache class provides an interface that provides the ability to create a cache to store messages in. Temporary offline SmartSockets clients will be able to have SmartSockets Cache store messages and retrieve them when they come back online.

This class is represented in the C API as TipcSrvConnCache.

- Methods**
- GMDFinish Method
  - GMDFlush Method
  - GMDInit Method
  - Init Method
  - RequestMsgs Method
  - RequestStart Method
  - RequestStop Method

- Properties**
- GroupNames Property
  - LbMode Property
  - RequestTimeout Property

**Events** None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## TipcSrvConnCache Methods

---

### GMDFinish Method

---

The GMDFinish method performs required clean up procedures when SmartSockets Cache stops to cache GMD message for this RTclient.

**Syntax** *object*.GMDFinish *cache\_subject*  
 where:  
*object* is an object expression that evaluates to a TipcSrvConnCache object.  
*cache\_subject* is a string that is the subject of SmartSockets Cache group.

**Return Type** None

**C API** TipcSrvConnCacheGMDFinish

**Example** This example calls the GmdFinish method:

```
Dim cache as New TipcSrvConnCache
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
Cache.Init SrvConn
Cache.GmdFinish "/smartcache"
```

### GMDFlush Method

---

The GMDFlush method flushes the SmartSockets Cache for cached messages from senders.

**Syntax** *object*.GMDFlush *request\_subject*, *request\_sender*, *cache\_subject*  
 where:  
*object* is an object expression that evaluates to a TipcSrvConnCache object.  
*request\_subject* is a string that is the Subject to be flushed.  
*request\_sender* is a string that is the Sender to be flushed.  
*cache\_subject* is a string that is the subject of SmartSockets Cache group.

|                    |                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Return Type</b> | None                                                                                                                                                                                                                                                                  |
| <b>C API</b>       | TipcSrvConnCacheGMDFlush                                                                                                                                                                                                                                              |
| <b>Example</b>     | <p>This example calls the GMDFlush method:</p> <pre>Dim cache as New TipcSrvConnCache Dim svc as New TipcSvc Dim SrvConn as TipcSrvConn Set SrvConn = svc.TipcSrvConnCreate("", "", "", "") Cache.Init SrvConn Cache.GmdFlush "/Foo", "/sender1", "/smartcache"</pre> |

## GMDInit Method

---

The GMDInit method performs initializations required to use SmartSockets Cache to cache GMD messages from this RTclient.

**Syntax**    *object.GMDInit cache\_subject*

where:

*object*            is an object expression that evaluates to a TipcSrvConnCache object.

*cache\_subject*    is a string that is the subject of SmartSockets Cache group.

|                    |                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Return Type</b> | None                                                                                                                                                                                                                                            |
| <b>C API</b>       | TipcSrvConnCacheGMDInit                                                                                                                                                                                                                         |
| <b>Example</b>     | <p>This example calls the GMDInit method:</p> <pre>Dim cache as New TipcSrvConnCache Dim svc as New TipcSvc Dim SrvConn as TipcSrvConn Set SrvConn = svc.TipcSrvConnCreate("", "", "", "") Cache.Init SrvConn Cache.GmdInit "/smartcache"</pre> |

## Init Method

The Init method performs initializations required to use SmartSockets Cache.

**Syntax** *object*.Init *SrvConn*

where:

*object* is an object expression that evaluates to a TipcSrvConnCache object.

*SrvConn* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** None

**C API** TipcSrvConnCacheInit

**Example** This example initializes the TipcSrvConnCache object with a server connection:

```
Dim cache as New TipcSrvConnCache
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
Cache.Init SrvConn
```

## RequestMsgs Method

The RequestMsgs method requests messages from SmartSockets Cache.

**Syntax** *object*.RequestMsgs *request\_subject*, *num\_msgs*, *delta\_time*, *cache\_subject*

where:

*object* is an object expression that evaluates to a TipcSrvConnCache object.

*request\_subject* is a string that is the requested subject; it can be a wildcard value.

*num\_msgs* is a long integer that is the number of messages.

*delta\_time* is a double real number that is the time span of messages.

*cache\_subject* is a string that is subject of SmartSockets Cache group.

**Return Type** None

**C API** TipcSrvConnCacheRequestMsgs

**Example** This example requests messages from the cache:

```
Dim cache as New TipcSrvConnCache
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
Cache.Init SrvConn
Cache.RequestMsgs "/Foo", 10, 30, "/smartcache"
```

## RequestStart Method

The RequestStart method sends a request to SmartSockets Cache to begin caching messages for a particular subject.

**Syntax** *object.RequestStart request\_subject, num\_msgs, delta\_time, cache\_subject*

where:

*object* is an object expression that evaluates to a TipcSrvConnCache object.

*request\_subject* is a string that is the requested subject, it can be a wildcard value

*num\_msgs* is a long integer that is the number of messages.

*delta\_time* is a double real number that is the time span of messages.

*cache\_subject* is a string that is subject of SmartSockets Cache group.

**Return Type** None

**C API** TipcSrvConnCacheRequestStart

**Example** This example does requests the cache to start caching on the "/Foo" subject:

```
Dim cache as New TipcSrvConnCache
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
Cache.Init SrvConn
Cache.RequestStart "/Foo", 10, 30, "/smartcache"
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## RequestStop Method

---

The RequestStop method sends a request to SmartSockets Cache to stop caching messages for a particular subject.

**Syntax** `object.RequestStop request_subject, cache_subject`

where:

*object* is an object expression that evaluates to a TipcSrvConnCache object.

*request\_subject* is a string that is the requested subject; it can be a wildcard value.

*cache\_subject* is a string that is subject of SmartSockets Cache group.

**Return Type** None

**C API** TipcSrvConnCacheRequestStop

**Example** This example calls the request stop method:

```
Dim cache as New TipcSrvConnCache
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
Cache.Init SrvConn
Cache.RequestStart "/Foo", 10, 30, "/smartcache"
Cache.RequestStop "/Foo", "/smartcache"
```

## TipcSrvConnCache Properties

### GroupNames Property

The GroupNames property sets or returns the names of the cache groups that this client will be requesting data from.

**Syntax**     *object*.GroupNames [=group\_names]  
              where:  
              *object*            is an object expression that evaluates to a TipcSrvConnCache object.  
              *group\_names* is a string that specifies the names of the cache groups that this client will be requesting data from. Most client API calls take a group name as an argument. If this parameter is specified as T\_NULL, the client library will use all of the cache group names listed in this option.

**Data Type**     String

**C API**            Cache\_Group\_Names option

**Example**        This example sets and gets the client's cache group names:

```
Dim cache as New TipcSrvConnCache
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
Cache.Init SrvConn
Cache.GroupNames = "/SmartCache1,/SmartCache2"
Debug.Print Cache.GroupNames
```

### LbMode Property

In publish-subscribe operations, the (load balancing mode) property specifies to which RTclient a message is routed.

If the message load balancing mode is not set, the message type load balancing mode is used. The default value for the message type load balancing is tipcLbNone.



|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>    | <p><code>object.LbMode [=lb_mode]</code></p> <p>where:</p> <p><code>object</code> is an object expression that evaluates to a <code>TipcMsg</code> object.</p> <p><code>lb_mode</code> is an integer that specifies the load balancing mode. Valid values are:</p> <ul style="list-style-type: none"><li>• 0 is for <code>tipcLbNone</code>, the default setting. The message is routed with no load balancing. The message is routed to all RTclients subscribing to the destination subject.</li><li>• 1 is for <code>tipcLbRoundRobin</code>. The message is routed to each RTclient subscribing to the destination subject, in turn.</li><li>• 2 is for <code>tipcLbWeighted</code>. The message is routed to the RTclient with the fewest number of unacknowledged messages.</li><li>• 3 is for <code>tipcLbSorted</code>. The message is routed to the RTclient in the lowest lexicographical order (sorted by unique subject).</li></ul> |
| <b>Data Type</b> | <code>TipcLbMode</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Remarks</b>   | See the <i>TIBCO SmartSockets User's Guide</i> for detailed information about load balancing.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>C API</b>     | <code>Cache_Lb_Mode</code> option                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Example</b>   | <p>This example changes a message's load balancing mode to round-robin:</p> <pre>'Change Load Balancing Mode to Round Robin Dim cache as New TipcSrvConnCache Dim svc as New TipcSvc Dim SrvConn as TipcSrvConn Set SrvConn = svc.TipcSrvConnCreate("", "", "", "") Cache.Init SrvConn cache.LbMode = tipcLbRoundRobin</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

## RequestTimeout Property

The `RequestTimeout` property sets or gets the number of seconds the client will wait for SmartSockets Cache to respond to a cache request.

|               |                                                                                                                                                                                                                                                                                                                                                                 |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b> | <p><code>object.RequestTimeout [=request_timeout]</code></p> <p>where:</p> <p><code>object</code> is an object expression that evaluates to a <code>TipcSrvConnCache</code> object.</p> <p><code>request_timeout</code> is a double real number specifying the number of seconds the client will wait for SmartSockets Cache to respond to a cache request.</p> |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                  |                                                                                                                                                                                                                                                                 |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Data Type</b> | Double                                                                                                                                                                                                                                                          |
| <b>C API</b>     | Cache_Request_Timeout option                                                                                                                                                                                                                                    |
| <b>Example</b>   | <p>This example does sets the RequestTimeout to 10 seconds:</p> <pre>Dim cache as New TipcSrvConnCache Dim svc as New TipcSvc Dim SrvConn as TipcSrvConn Set SrvConn = svc.TipcSrvConnCreate("", "", "", "") Cache.Init SrvConn Cache.RequestTimeout = 10</pre> |

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## TipcSrvMon Class

---

TipcSrvMon — class for monitoring SmartSockets applications.

**Description** The TipcSrvMon class is used to monitor SmartSockets applications.

**Methods**

ClientBufferGetWatch Method  
 ClientBufferPoll Method  
 ClientBufferSetWatch Method  
 ClientCbPoll Method  
 ClientCongestionGetWatch Method  
 ClientCongestionSetWatch Method  
 ClientCpuPoll Method  
 ClientExtPoll Method Method  
 ClientGeneralPoll Method  
 ClientInfoPoll Method  
 ClientMsgRecvGetWatch Method  
 ClientMsgRecvSetWatch Method  
 ClientMsgSendGetWatch Method  
 ClientMsgSendSetWatch Method  
 ClientMsgTrafficPoll Method  
 ClientMsgTypePoll Method  
 ClientNamesGetWatch Method  
 ClientNamesNumPoll Method  
 ClientNamesPoll Method  
 ClientNamesSetWatch Method  
 ClientOptionPoll Method  
 ClientSubjectPoll Method  
 ClientSubscribeGetWatch Method  
 ClientSubscribeNumPoll Method  
 ClientSubscribePoll Method  
 ClientSubscribeSetWatch Method  
 ClientVersionPoll  
 Create Method  
 Destroy Method  
 ProjectNamesGetWatch Method  
 ProjectNamesPoll Method  
 ProjectNamesSetWatch Method

ServerBufferPoll Method  
 ServerCongestionGetWatch Method  
 ServerCongestionSetWatch Method  
 ServerConnGetWatch Method  
 ServerConnPoll Method  
 ServerConnSetWatch Method  
 ServerCpuPoll Method  
 ServerGeneralPoll Method  
 ServerMaxClientLicensesGetWatch Method  
 ServerMaxClientLicensesSetWatch Method  
 ServerMsgTrafficPoll Method  
 ServerNamesGetWatch Method  
 ServerNamesPoll Method  
 ServerNamesSetWatch Method  
 ServerOptionPoll Method  
 ServerRoutePoll Method  
 ServerTimePoll Method  
 ServerStartTimePoll  
 ServerVersionPoll  
 SubjectNamesGetWatch Method  
 SubjectNamesPoll Method  
 SubjectNamesSetWatch Method  
 SubscribeGetWatch Method  
 SubscribePoll Method  
 SubscribeSetWatch Method

**Properties and  
Events**

None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## TipcSrvMon Class Methods

### ClientBufferGetWatch Method

The ClientBufferGetWatch method determines if this RTclient is watching message-related buffer information for other RTclients.

**Syntax**     *object*.ClientBufferGetWatch (*client\_name*, *watch\_status*)

where:

*object*             is an object expression that evaluates to a TipcSrvMon object.

*client\_name*       is a string that specifies the name of RTclient for determining watch status. Wildcard names are allowed, or use "/"... to indicate a watch for all RTclients matching the MonitorScope property.

*watch\_status*     is a Boolean value representing the watch status:

- True means that watching is active
- False means there is no watching activity

**Return Type**     None

**C API**             TipcSrvMonClientBufferGetWatch

**Example**           This example determines if RTclient is watching all clients for message-related buffer information:

```
Dim status As Boolean
Dim Monitor as new TipcSrvMon
Dim svc as New TipcSvc
Dim conn as TipcSrvConn
Set conn = svc.TipcSrvConnCreate("", "", "", "")
Monitor.Init conn

Monitor.ClientBufferGetWatch("/...", status)
If status = False Then
 Debug.Print "Client Buffer not being watched"
Else
 Debug.Print "Client Buffer being watched"
End If
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## ClientBufferPoll Method

The ClientBufferPoll method polls once for RTclient message-related buffer information.

**Syntax** `object.ClientBufferPoll (client_name)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/"..." to indicate a poll of all RTclients matching the MonitorScope property.

**Return Type** None

**C API** TipcSrvMonClientBufferPoll

**Example** This example polls for RTclient buffer information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
Monitor.Init conn

'Poll all clients including self
Monitor.ClientBufferPoll "/"..."
SrvConn.Flush

'Process the poll message sent to self first
Set mt = Svc.MtLookup("mon_client_buffer_poll_call")
Set msg = SrvConn.MsgSearchType(10.0, mt)
If Not msg is Nothing Then
 SrvConn.MsgProcess msg
 msg.Destroy
End If

'Process the poll result message
Set mt = Svc.MtLookup("mon_client_buffer_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not msg is Nothing
 Debug.Print "RTclient name = " & msg.NextStr
 Debug.Print "msg queue count = " & msg.NextInt4
 Debug.Print "msg queue byte count = " & msg.NextInt4
 Debug.Print "read buffer count = " & msg.NextInt4
 Debug.Print "write buffer count = " & msg.NextInt4
 msg.Destroy
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend
```

## ClientBufferSetWatch Method

The ClientBufferSetWatch method starts or stops the watching process for an RTclient regarding message-related buffer information.

**Syntax**     *object*.ClientBufferSetWatch (*client\_name*, *watch\_status*)

where:

*object*        is an object expression that evaluates to a TipcSrvMon object.

*client\_name*   is a string that specifies the name of RTclient for setting the watch status. Wildcard names are allowed, or use "/..." to indicate a watch of all RTclients matching the MonitorScope property.

*watch\_status* is a Boolean value that sets the watch status:

- True starts the watching process
- False stops the watching process

**Return Type**     None

**C API**            TipcSrvMonClientBufferSetWatch

**Example**        This example starts watching for message-related buffer information for all RTclients. A callback is registered to handle the status message:

```
Dim mt As TipcMt
Monitor as new TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init conn

set mt = svc.MtLookup("mon_client_buffer_status")
SrvConn.ProcessCbCreate process_cb, mt, 1, Nothing
Monitor.ClientBufferSetWatch("/...", True)
SrvConn.Flush
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED



## ClientCbPoll Method

The ClientBufferPoll method polls once for RTclient callback information.

**Syntax** `object.ClientCbPoll (client_name)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

**Return Type** None

**C API** TipcSrvMonClientCbPoll

**Example** This example polls for RTclient callback information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init conn

'Poll all clients including self
Monitor.ClientCbPoll "/"...
SrvConn.Flush

'Process the poll message sent to self first
Set mt = Svc.MtLookup("mon_client_cb_poll_call")
Set msg = SrvConn.MsgSearchType(10.0, mt)
If Not Msg is Nothing Then
 SrvConn.MsgProcess msg
 msg.Destroy
End If

Set mt = Svc.MtLookup("mon_client_cb_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 Debug.Print "RTclient name = " & msg.NextStr
 Debug.Print "num global read cb = " & msg.NextInt4
 Debug.Print "num global write cb = " & msg.NextInt4
 Debug.Print "num global process cb = " & msg.NextInt4
 Debug.Print "num global queue cb = " & msg.NextInt4
 Debug.Print "num default cb = " & msg.NextInt4
 Debug.Print "num error cb = " & msg.NextInt4
 Debug.Print "num server create cb= " & msg.NextInt4
 Debug.Print "num server destroy cb = " & msg.NextInt4
 msg.Destroy
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend
```

## ClientCongestionGetWatch Method

The ClientCongestionGetWatch method determines if this RTclient is watching an RTclient write buffer.

**Syntax**     *object*.ClientCongestionGetWatch (*client\_name*, *watch\_status*)

where:

*object*             is an object expression that evaluates to a TipcSrvMon object.

*client\_name*       is a string that specifies the name of RTclient for determining watch status. Wildcard names are allowed, or use "/"..." to indicate a watch for all RTclients matching the MonitorScope property.

*watch\_status*      is a Boolean value representing the watch status:

- True means watching is active.
- False means there is no watching activity.

**Return Type**     None

**C API**            TipcSrvMonClientCongestionGetWatch

**Example**          This example determines if this RTclient is watching for write buffer congestion:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ClientCongestionGetWatch("/...", status)
If result = True Then
 If status = False Then
 Debug.Print "Client Congestion not being watched"
 Else
 Debug.Print "Client Congestion being watched"
 End If
Else
 Debug.Print "ClientCongestionGetWatch failed!"
End If
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## ClientCongestionSetWatch Method

The ClientCongestionSetWatch method starts or stops the watching of RTclient write buffer congestion.

**Syntax** `object.ClientCongestionSetWatch (client_name, high_water, low_water, watch_status)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of RTclient for determining watch status. Wildcard names are allowed, or use "/"... to indicate a watch for all RTclients matching the MonitorScope property.

*high\_water* the number of pending bytes that must be in the 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* is a Boolean value that sets the watch status:

- True starts the watching process.
- False stops the watching process.

**Return Type** None

**C API** TipcSrvMonClientCongestionSetWatch

**Example** This example sets a watch for write buffer congestion:

```
Dim mt As TipcMt
Dim result As Boolean

mt.Lookup "mon_client_congestion_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ClientCongestionSetWatch("/...", 1, 10, True)
TipcSvc1.Srv.Flush
If result = False Then
 Debug.Print "ClientCongestionSetWatch failed!"
End If
```

## ClientCpuPoll Method

The ClientCpuPoll method polls for an RTclient's CPU percentage used.

**Syntax** `object.ClientCpuPoll (client_name)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient to poll for CPU use. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

**Return Type** None

**C API** TipcSrvMonClientCpuPoll

**Example** This example polls for RTclient CPU usage:

```
Dim mt As TipcMt
Dim msg As TipcMsg

'Poll all clients including self
TipcMon1.ClientCpuPoll "/"...
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_cpu_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10#, mt)
If msg.IsNull = False Then
 TipcSvc1.Srv.MsgProcess msg
End If

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_client_cpu_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10#, mt)
While msg.IsNull = False
 On Error GoTo EndOfMessage
 eom = False
 While Not eom

 Debug.Print "Client name: " & msg.NextStr
 Debug.Print "CPU Utilization: " & msg.NextInt4

 Set msg = TipcSvc1.Srv.MsgSearchType(10#, mt)
 Wend
EndOfMessage:
 eom = True
Exit Sub
Wend
```

## ClientExtPoll Method

The ClientExtPoll method polls for extension data of an RTclient. 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.

See the TipcSrvMonExt Class Methods on page 520 for the methods to define, set, or delete fields of extension data in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax** `object.ClientExtPoll (client_name)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is the name of the RTclient to retrieve extension data from. The value of *client\_name* can be:

- the unique subject name of an RTclient
- a subject specified with wildcard characters to match multiple RTclients; for example, /stocks/auto/... or /stocks1a\*/...
- T\_IPC\_MON\_ALL which specifies to poll all RTclients in the project that match the value of the Monitor\_Scope option

**Return Type** None

**C API** TipcSrvMonClientExtPoll

**Example** This example polls for RTclient extension data:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim Monitor As TipcSrvMon
Dim MonExt As TipcSrvMonExt
Dim svc As New TipcSvc
Dim SrvConn As TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
Set Monitor = SrvConn.TipcSrvMonCreate()
Set MonExt = SrvConn.TipcSrvMonExtCreate()

MonExt.SetStr "id", "foo"

'Poll all clients including self
Monitor.ClientExtPoll "/..."
SrvConn.Flush

'Process the poll message sent to self first
Set mt = svc.MtLookup("mon_client_extension_poll_call")
Set msg = SrvConn.MsgSearchType(10#, mt)
If Not msg Is Nothing Then
 SrvConn.MsgProcess msg
End If
```

```

'Process the poll result message
Set mt = svc.MtLookup("mon_client_extension_poll_result")
Set msg = SrvConn.MsgSearchType(10#, mt)
While Not msg Is Nothing
 Debug.Print "RTclient name = " & msg.NextStr
 Debug.Print "RTclient id = " & msg.GetNamedStr("id")
 Set msg = SrvConn.MsgSearchType(10#, mt)
Wend

```

## ClientGeneralPoll Method

The ClientGeneralPoll method polls once for RTclient general information.

**Syntax** `object.ClientGeneralPoll (client_name)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

**Return Type** None

**C API** TipcSrvMonClientGeneralPoll

**Example** This example polls for RTclient general information:

```

Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim i_array() As Long
Dim s_array() As String
Dim Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all clients including self
Monitor.ClientGeneralPoll "/"...
SrvConn.Flush

'Process the poll message sent to self first
Set mt = Svc.MtLookup("mon_client_general_poll_call")
Set msg = SrvConn.MsgSearchType(10.0, mt)
If Not msg Is Nothing Then
 SrvConn.MsgProcess msg
End If

Set mt = Svc.MtLookup("mon_client_general_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not msg Is Nothing

```

```

Debug.Print "RTclient: " & msg.NextStr
Debug.Print "Ident: " & msg.NextStr
Debug.Print "Node: " & msg.NextStr
Debug.Print "User: " & msg.NextStr
Debug.Print "Pid: " & msg.NextInt4
Debug.Print "Project: " & msg.NextStr
Debug.Print "Server: " & msg.NextStr
Debug.Print "Architecture: " & msg.NextStr
Debug.Print "Current SBRK: " & msg.NextInt4
Debug.Print "Delta SBRK: " & msg.NextInt4
Debug.Print "Integer format: " & msg.NextInt2
Debug.Print "Real format: " & msg.NextInt2
Debug.Print "LCN: " & msg.NextStr

Debug.Print "Counted licenses:"
msg.NextInt4Array(i_array, numItems)
For i = 0 To numItems - 1
 Debug.Print " " & i_array(i)
Next i

Debug.Print "Extra licenses:"
msg.NextStrArray(s_array, numItems)
For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
Next i

Debug.Print "Subjects:"
msg.NextStrArray(s_array, numItems)
For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
Next i

msg.Destroy
Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend

```

## ClientInfoPoll Method

The ClientInfoPoll method polls once for RTclient information.

**Syntax**     *object*.ClientInfoPoll (*client\_name*)

where:

*object*        is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient to poll. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

**Return Type**     None

**C API**        TipcSrvMonClientInfoPoll

**Example** This example polls for RTclient information:

```

Dim mt As TipcMt
Dim msg As TipcMsg

'Poll all clients including self
TipcMon1.ClientInfoPoll "/"...
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_info_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10#, mt)
If msg.IsNull = False Then
 TipcSvc1.Srv.MsgProcess msg
End If

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_client_info_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10#, mt)
While msg.IsNull = False
 On Error GoTo EndOfMessage
 eom = False
 While Not eom
 Debug.Print "Client name: " & msg.NextStr
 Debug.Print "Ident: " & msg.NextStr
 Debug.Print "Node name: " & msg.NextStr
 Debug.Print "User name: " & msg.NextStr
 Debug.Print "PID: " & msg.NextInt4
 Debug.Print "Project: " & msg.NextStr
 Debug.Print "Server name: " & msg.NextStr
 Debug.Print "Arch: " & msg.NextStr
 Debug.Print "Current sbrk: " & msg.NextInt4
 Debug.Print "sbrk delta since startup: " & msg.NextInt4
 Debug.Print "int format: " & msg.NextInt2
 Debug.Print "real format: " & msg.NextInt2
 Debug.Print "logical connection name to the server: " &
 msg.NextStr
 Debug.Print "No of subscribes for client: " & msg.NextInt4
 Debug.Print "% CPU Utilization: " & msg.NextReal4

 Set msg = TipcSvc1.Srv.MsgSearchType(10#, mt)
 Wend
EndOfMessage:
 eom = True
 Exit Sub
Wend

```



## ClientMsgRecvGetWatch Method

The ClientMsgRecvGetWatch method determines if this RTclient is watching received messages in an RTclient.

**Syntax** `object.ClientMsgRecvGetWatch (client_name, msg_type, watch_status)`  
 where:  
*object* is an object expression that evaluates to a TipcSrvMon object.  
*client\_name* is a string that specifies the name of the RTclient for determining watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTclients matching the MonitorScope property.  
*msg\_type* is a string that specifies the name of the message for getting watch status. Use "@" to indicate a watch of all message types.  
*watch\_status* is a Boolean value representing the watch status:

- True means that watching is active.
- False means there is no watching activity.

**Return Type** None

**C API** TipcSrvMonClientMsgRecvGetWatch

**Example** This example determines if this RTclient is watching all message types for the current RTclient:

```
Dim status As Boolean
Dim Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

Monitor.ClientMsgRecvGetWatch SrvConn.Unique_Subject, "@", status
If status = False Then
 Debug.Print "Messages received not being watched"
Else
 Debug.Print "Messages received being watched"
End If
```

## ClientMsgRecvSetWatch Method

The ClientMsgRecvSetWatch method starts or stops watching for messages received for RTclient.

**Syntax**     *object*.ClientMsgRecvSetWatch (*client\_name*, *msg\_type*, *watch\_status*)

where:

*object*        is an object expression that evaluates to a TipcSrvMon object.

*client\_name*   is a string that specifies the name of the RTclient for setting the watch status. Wildcard names are allowed, or use "/. . ." to indicate a watch of all RTclients matching the MonitorScope property.

*msg\_type*      is a string that specifies the name of the message for setting the watch status. Use "@" to indicate a watch of all message types.

*watch\_status* is a Boolean value that sets the watch status:

- True starts the watching process.
- False stops the watching process.

**Return Type**     None

**C API**            TipcSrvMonClientMsgRecvSetWatch

**Example**        This example starts watching for messages received by the current client. A callback is registered to handle the status message:

```
Dim mt As TipcMt
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

set mt = svc.MtLookup("mon_client_msg_recv_status"
SrvConn.ProcessCbCreate process_cb, mt, 1, Nothing
Monitor.ClientMsgRecvSetWatch SrvConn.Unique_Subject, "@", True
SrvConn.Flush
If result = False Then
 Debug.Print "ClientMsgRecvSetWatch failed!"
End If
```

## ClientMsgSendGetWatch Method

The ClientMsgSendGetWatch method determines if this RTclient is watching sent messages in an RTclient.

**Syntax** `object.ClientMsgSendGetWatch (client_name, msg_type, watch_status)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient for determining watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTclients matching the MonitorScope property.

*msg\_type* is a string that specifies the name of the message for retrieving watch status. Use "@" to indicate a watch of all message types.

*watch\_status* is a Boolean value representing the watch status:

- True means that watching is active.
- False mean there is no watching activity.

**Return Type** None

**C API** TipcSrvMonClientMsgSendGetWatch

**Example** This example determines if sent messages are watched for the current RTclient:

```
Dim status As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

Monitor.ClientMsgSendGetWatch SrvConn.Unique_Subject, "@", status
If status = False Then
 Debug.Print "Messages sent not being watched"
Else
 Debug.Print "Messages sent being watched"
End If
```

## ClientMsgSendSetWatch Method

The ClientMsgSendSetWatch method starts or stops the watching of messages sent by an RTclient.

**Syntax**     *object*.ClientMsgSendSetWatch (*client\_name*, *msg\_type*, *watch\_status*)

where:

*object*        is an object expression that evaluates to a TipcSrvMon object.

*client\_name*   is a string that specifies the name of the RTclient for setting the watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTclients matching the MonitorScope property.

*msg\_type*      is a string that specifies the name of the message for setting the watch status. Use "@" to indicate a watch of all message types.

*watch\_status* is a Boolean value that sets the watch status:

- True starts the watching process.
- False stops the watching process.

**Return Type**     Boolean

**C API**            TipcSrvMonClientMsgSendSetWatch

**Example**        This example starts watching for messages sent by the current client. A callback is registered to handle the status message:

```
Dim mt As TipcMt
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn
set mt = svc.MtLookup("mon_client_msg_send_status")
SrvConn.ProcessCbCreate process_cb, mt, 1, Nothing
Monitor.ClientMsgSendSetWatch SrvConn.UniqueSubject, "@", True
SrvConn.Flush
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## ClientMsgTrafficPoll Method

The ClientMsgTrafficPoll method polls once for RTclient message traffic.

**Syntax** `object.ClientMsgTrafficPoll (client_name)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

**Return Type** None

**C API** TipcSrvMonClientMsgTrafficPoll

**Example** This example polls for RTclient message traffic information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all clients including self
Monitor.ClientMsgTrafficPoll "/"...
SrvConn.Flush

'Process the poll message sent to self first
Set mt = Svc.MtLookup("mon_client_msg_traffic_poll_call")
Set msg = SrvConn.MsgSearchType(10.0, mt)
If Not Msg is Nothing Then
 SrvConn.MsgProcess msg
End If

'Process the poll result message
Set mt = Svc.MtLookup("mon_client_msg_traffic_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 Debug.Print "RTclient name = " & msg.NextStr
 Debug.Print "total msgs send= " & msg.NextInt4
 Debug.Print "total msgs recv= " & msg.NextInt4
 Debug.Print "total bytes send = " & msg.NextInt4
 Debug.Print "total bytes recv = " & msg.NextInt4
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend
```

## ClientMsgTypePoll Method

The ClientMsgTypePoll method polls once for RTclient message type information.

**Syntax** `object.ClientMsgTypePoll (client_name, msg_type)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/" . . ." to indicate a poll of all RTclients matching the MonitorScope property.

*msg\_type* is a string that specifies the name of message type to poll for information. Use "@" to indicate all message types.

**Return Type** None

**C API** TipcSrvMonClientMsgTypePoll

**Example** This example polls for RTclient message type information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all clients including self
Monitor.ClientMsgTypePoll SrvConn.UniqueSubject, "@"
SrvConn.Flush

'Process the poll message sent to self first
Set mt = Svc.MtLookup("mon_client_msg_type_poll_call")
Set msg = SrvConn.MsgSearchType(10.0, mt)
If Not Msg is Nothing Then
 SrvConn.MsgProcess msg
 msg.Destroy
End If

Set mt = Svc.MtLookup("mon_client_msg_type_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 On Error Goto EndOfMessage
 eom = False
 Debug.Print "RTclient:" & msg.NextStr
 While Not eom
 Debug.Print " msg_type: " & msg.NextStr
 Debug.Print " number: " & msg.NextInt4
 Debug.Print " grammar: " & msg.NextStr
 Debug.Print " priority_known: " & msg.NextInt4
 Debug.Print " priority: " & msg.NextInt2
 Debug.Print " delivery_mode: " & msg.NextInt4
 Debug.Print " user property: " & msg.NextInt4
 Debug.Print " num read cb: " & msg.NextInt4
```

```

 Debug.Print " num write cb: " & msg.NextInt4
 Debug.Print " num process cb: " & msg.NextInt4
 Debug.Print " num queue cb: " & msg.NextInt4
 Debug.Print " total msg send: " & msg.NextInt4
 Debug.Print " total msg recv: " & msg.NextInt4
 Debug.Print " total byte send: " & msg.NextInt4
 Debug.Print " total byte recv: " & msg.NextInt4
 Wend
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend
EndOfMessage:
 eom = True
Exit Sub

```

## ClientNamesGetWatch Method

The ClientNamesGetWatch method determines if this RTclient is watching RTclient names.

**Syntax** *object*.ClientNamesGetWatch (*watch\_status*)

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*watch\_status* is a Boolean value representing the watch status:

- True means that watching is active.
- False means there is no watching activity.

**Return Type** None

**C API** TipcSrvMonClientNamesGetWatch

**Example** This example determines if an RTclient is watching client names:

```

Dim status As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

Monitor.ClientNamesGetWatch(status)
If status = False Then
 Debug.Print "Client names not being watched"
Else
 Debug.Print "Client names being watched"
End If

```

## ClientNamesNumPoll Method

---

The ClientNamesNumPoll method polls for the number of RTclients in the current project.

**Syntax**     *object*.ClientNamesNumPoll  
where *object* is an object expression that evaluates to a TipcSrvMon object.

**Return Type**     None

**C API**     TipcSrvMonClientNamesNumPoll

**Example**     This example polls for the number of RTclients:

```
Dim mt As TipcMt
Dim msg As TipcMsg

'Poll all clients including self
TipcMon1.ClientNamesNumPoll
TipcSvc1.Srv.Flush

'Process the poll result message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_names_num_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)

While msg.IsNull = False
 Debug.Print "No of Clients:"
 'Find the no of clients.
 Debug.Print " " & msg.NextInt4

 Set msg = TipcSvc1.Srv.MsgSearchType(10, mt)
Wend
```

## ClientNamesPoll Method

---

The ClientNamesPoll method polls once for RTclient names.

**Syntax**     *object*.ClientNamesPoll  
where *object* is an object expression that evaluates to a TipcSrvMon object.

**Return Type**     None

**C API**     TipcSrvMonClientNamesPoll



**Example** This example polls for RTclient names:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array As String
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all clients including self
Monitor.ClientNamesPoll
SrvConn.Flush

'Process the poll result message sent to self first
Set mt = Svc.MtLookup("mon_client_names_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 Debug.Print "Client names:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Debug.Print "Client info strings:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Debug.Print "server names:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend
```

## ClientNamesSetWatch Method

The ClientNamesSetWatch method starts or stops the watching of RTclient names.

**Syntax** *object*.ClientNamesSetWatch (*watch\_status*)

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*watch\_status* is a Boolean value that sets the watch status:

- True starts the watching process.
- False stops the watching process.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Return Type</b> | None                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>C API</b>       | TipcSrvMonClientNamesSetWatch                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Example</b>     | <p>This example starts watching for RTclient names. A callback is registered to handle the status message:</p> <pre>Dim mt As TipcMt Monitor as New TipcSrvMon Dim svc as New TipcSvc Dim SrvConn as TipcSrvConn Set SrvConn = svc.TipcSrvConnCreate("", "", "", "") SrvConn.Open Monitor.Init SrvConn  set mt = svc.MtLookup( "mon_client_names_status" SrvConn.ProcessCbCreate process_cb, mt, 1, Nothing Monitor.ClientNamesSetWatch(True) SrvConn.Flush</pre> |

## ClientOptionPoll Method

The ClientOptionPoll method polls once for RTclient options.

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b> | <p><i>object</i>.ClientOptionPoll (<i>client_name</i>, <i>option_name</i>)</p> <p>where:</p> <p><i>object</i> is an object expression that evaluates to a TipcSrvMon object.</p> <p><i>client_name</i> is a string that specifies the name of RTclient for polling. Wildcard names are allowed, or use "/. . ." to indicate a poll of all RTclients matching the MonitorScope property.</p> <p><i>option_name</i> is a string that specifies the name of the option to poll for information. Use "@" to indicate a poll of all options.</p> |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                    |                            |
|--------------------|----------------------------|
| <b>Return Type</b> | None                       |
| <b>C API</b>       | TipcSrvMonClientOptionPoll |

**Example** This example polls for RTclient option information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean
Dim numItems As Long
Dim i As Long
Dim s_array As String
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

```

SrvConn.Open
Monitor.Init SrvConn

'Poll all clients including self
Monitor.ClientOptionPoll "/...", "@"
SrvConn.Flush

'Process the poll message sent to self first
Set mt = Svc.MtLookup("mon_client_option_poll_call")

Set msg = SrvConn.MsgSearchType(10.0, mt)
If Not Msg is Nothing Then
 SrvConn.MsgProcess msg
 msg.Destroy
End If

'Process the poll result message
Set mt = Svc.MtLookup("mon_client_option_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 On Error Goto EndOfMessage
 eom = False
 Debug.Print "RTclient:" & msg.NextStr
 While Not eom
 Debug.Print " option: " & msg.NextStr
 Debug.Print " type: " & msg.NextInt2
 Debug.Print " value: " & msg.NextStr
 Debug.Print " required: " & msg.NextInt4
 Debug.Print " legal values:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i
 Wend
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend

EndOfMessage:
eom = True
Exit Sub

```

## ClientSubjectPoll Method

The ClientSubjectPoll method polls once for the statistics associated with the RTclient subscriptions.

**Syntax** *object*.ClientSubjectPoll (*client\_name*, *subject*)

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient for polling. Wildcard names are allowed, or use "/"... to indicate a poll of all RTclients matching the MonitorScope property.

*subject* is a string that specifies the name of the subject to poll for information. Use "@" to indicate a poll of all subjects.

**Return Type**     None

**C API**            TipcSrvMonClientSubjectPoll

**Example**          This example polls for RTclient subject information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all clients including self
Monitor.ClientSubjectPoll "/...", "@"
SrvConn.Flush

'Process the poll message sent to self first
Set mt = Svc.MtLookup("mon_client_subject_poll_call")
Set msg = SrvConn.MsgSearchType(10.0, mt)
If Not Msg is Nothing Then
 SrvConn.MsgProcess msg
 msg.Destroy
End If

'Process the poll result message
Set mt = Svc.MtLookup("mon_client_subject_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 On Error Goto EndOfMessage
 eom = False
 Debug.Print "RTclient:" & msg.NextStr
 While Not eom
 Debug.Print " Subject = " & msg.NextStr
 Debug.Print " total msgs send = " & msg.NextInt4
 Debug.Print " total msgs recv = " & msg.NextInt4
 Debug.Print " total bytes send = " & msg.NextInt4
 Debug.Print " total bytes recv = " & msg.NextInt4
 Wend
 msg.Destroy
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend
EndOfMessage:
eom = True
Exit Sub
```

## ClientSubscribeGetWatch Method

The ClientSubscribeGetWatch method determines if this RTclient is watching the subjects to which an RTclient is subscribing.

**Syntax** `object.ClientSubscribeGetWatch (client_name, watch_status)`  
 where:  
*object* is an object expression that evaluates to a TipcSrvMon object.  
*client\_name* is a string that specifies the name of the RTclient for determining watch status. Wildcard names are allowed, or use "/"... to indicate a watch of all RTclients matching the MonitorScope property.  
*watch\_status* is a Boolean value representing the watch status:

- True means that watching is active.
- False means there is no watching activity.

**Return Type** None

**C API** TipcSrvMonClientSubscribeGetWatch

**Example** This example determines if the RTclient is watching the subjects to which an RTclient is subscribing:

```
Dim status As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

Monitor.ClientSubscribeGetWatch("/client1", status)
If status = False Then
 Debug.Print "Subjects not being watched"
Else
 Debug.Print "Subjects being watched"
End If
```

## ClientSubscribeNumPoll Method

The ClientSubscribeNumPoll method polls once for the number of subjects to which an RTclient is subscribing.

**Syntax** `object.ClientSubscribeNumPoll (client_name)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient to poll for the number of subject subscriptions. Wildcard names are allowed, or use `"/..."` to indicate a poll of all RTclients matching the MonitorScope property.

**Return Type** None

**C API** TipcSrvMonClientSubscribeNumPoll

**Example** This example polls all RTclients for the number of subjects to which they are subscribed:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim i As Long
Dim s_array(32) As String
Dim i_array(32) As Long

'Poll all clients including self
TipcMon1.ClientSubscribeNumPoll "/"...
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_client_subscribe_num_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
If msg.IsNull = False Then
 TipcSvc1.Srv.MsgProcess msg
End If

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_client_subscribe_num_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
While msg.IsNull = False
 On Error GoTo EndOfMessage
 eom = False
 While Not eom

 Debug.Print "Client names:"
 msg.NextStrArray s_array, 32
 For i = 0 To 31
 Debug.Print " " & s_array(i)
 Next i

 Debug.Print "No of subscribes per client:"
 msg.NextInt4Array i_array, 32
 For i = 0 To 31
 Debug.Print " " & i_array(i)
```

```

 Next i

 Set msg = TipcSvc1.Srv.MsgSearchType(10.0, mt)
 Wend

EndOfMessage:
 eom = True
Exit Sub

Wend

```

## ClientSubscribePoll Method

The ClientSubscribePoll method polls once for the subjects to which an RTclient is subscribing.

**Syntax** *object*.ClientSubscribePoll (*client\_name*)

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient to poll for the subject subscription. Wildcard names are allowed, or use "/"..." to indicate a poll of all RTclients matching the MonitorScope property.

**Return Type** None

**C API** TipcSrvMonClientSubscribePoll

**Example** This example polls all RTclients for the subjects to which they are subscribed:

```

Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array() As String
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all clients including self
Monitor.ClientSubscribePoll "/"..."
SrvConn.Flush

'Process the poll message sent to self first
Set mt = Svc.MtLookup("mon_client_subscribe_poll_call")
Set msg = SrvConn.MsgSearchType(10.0, mt)
If Not msg Is Nothing Then
 SrvConn.MsgProcess msg
End If

```

```
'Process the poll result message
Set mt = Svc.MtLookup("mon_client_subscribe_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 On Error Goto EndOfMessage
 eom = False
 While Not eom
 Debug.Print " RTclient subjects:" & msg.NextStr
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i
 Wend
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend

EndOfMessage:
eom = True
Exit Sub
```

## ClientSubscribeSetWatch Method

The ClientSubscribeSetWatch method starts or stops the watching of subjects to which an RTclient is subscribing.

**Syntax**

```
object.ClientSubscribeSetWatch (client_name, watch_status)
```

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*client\_name* is a string that specifies the name of the RTclient for setting watch status. Wildcard names are allowed, or use "/"..." to indicate a watch of all RTclients matching the MonitorScope property.

*watch\_status* is a Boolean value that sets the watch status:

- True starts the watching process.
- False stops the watching process.

**Return Type**    None

**C API**    TipcSrvMonClientSubscribeSetWatch

**Example**    This example starts watching for subjects to which an RTclient is subscribing. A callback is registered to handle the status message:

```
Dim mt As TipcMt
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn
```



```
set mt = svc.MtLookup("mon_client_subscribe_status"
SrvConn.ProcessCbCreate process_cb, mt, 1, Nothing
Monitor.ClientSubscribeSetWatch("/...", True)
SrvConn.Flush
```

## Create Method

The Create method creates the TipcSrvMon object and assigns the connection it will use to publish monitoring messages.

**Syntax** *object*.Create *SrvConn*

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*SrvConn* is an object expression that evaluates to a TipcSrvConn object.

**Return Type** None

**C API** No comparable C API function

**Example** This example creates the TipcSrvMon object:

```
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Create SrvConn
```

## Destroy Method

The Destroy method destroys the TipcSrvMon object.

**Syntax** *object*.Destroy

where *object* is an object expression that evaluates to a TipcSrvMon object.

**Return Type** None

**C API** No comparable C API function

**Example** This example destroys a monitor object:

```
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Create SrvConn
Monitor.Destroy
```

## ProjectNamesGetWatch Method

The ProjectNamesGetWatch method determines if this RTclient is watching project names.

**Syntax** *object*.ProjectNamesGetWatch (*watch\_status*)

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*watch\_status* is a Boolean value representing the watch status:

- True means that watching is active.
- False means there is no watching activity.

**Return Type** None

**C API** TipcSrvMonProjectNamesGetWatch

**Example** This example determines if this RTclient is watching project names:

```
Dim status As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

Monitor.ProjectNamesGetWatch(status)
If status = False Then
 Debug.Print "Project names not being watched"
Else
 Debug.Print "Project names being watched"
End If
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## ProjectNamesPoll Method

---

The ProjectNamesPoll method polls once for project names.

**Syntax** `object.ProjectNamesPoll`

where *object* is an object expression that evaluates to a TipcSrvMon object.

**Return Type** None

**C API** TipcSrvMonProjectNamesPoll

**Example** This example polls for RTclient names:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array() As String
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll RTserver for the list of project names
Monitor.ProjectNamesPoll
SrvConn.Flush

'Process the poll result message
Set mt = Svc.MtLookup("mon_project_names_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 Debug.Print "Project names:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend
```

## ProjectNamesSetWatch Method

The ProjectNamesSetWatch method starts or stops the watching of project names.

**Syntax**     *object*.ProjectNamesSetWatch (*watch\_status*)  
where:  
*object*        is an object expression that evaluates to a TipcSrvMon object.  
*watch\_status* is a Boolean value that sets the watch status:

- True starts the watching process.
- False stops the watching process.

**Return Type**     None

**C API**            TipcSrvMonProjectNamesSetWatch

**Example**        This example starts watching for project names. A callback is registered to handle the status message:

```
Dim mt As TipcMt
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

set mt = svc.MtLookup("mon_project_names_status")
SrvConn.ProcessCbCreate process_cb, mt, 1, Nothing
Monitor.ProjectNamesSetWatch(True)
SrvConn.Flush
```

## ServerBufferPoll Method

The ServerBufferPoll method polls once for RTserver message-related buffer information.

**Syntax**     *object*.ServerBufferPoll (*server\_name*, *client\_name*)  
where:  
*object*        is an object expression that evaluates to a TipcSrvMon object.  
*server\_name* is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/..." to indicate a poll of all RTservers matching the MonitorScope property.

*client\_name* is a string that specifies the name of the RTclient, connected to the RTserver, to poll. Use "@" to indicate a poll of all subjects.

**Return Type** None

**C API** TipcSrvMonServerBufferPoll

**Example** This example polls for RTserver message-related buffer information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all RTservers
Monitor.ServerBufferPoll "/...", "@"
SrvConn.Flush

'Process the poll result message
Set mt = Svc.MtLookup("mon_server_buffer_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 On Error Goto EndOfMessage
 eom = False
 Debug.Print "RTserver:" & msg.NextStr
 While Not eom
 Debug.Print " RTclient: " & msg.NextStr
 Debug.Print " Message Q count = " & msg.NextInt4
 Debug.Print " Message Q byte count = " & msg.NextInt4
 Debug.Print " read buffer count = " & msg.NextInt4
 Debug.Print " write buffer count = " & msg.NextInt4
 Wend
 msg.Destroy
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend
EndOfMessage:
eom = True
Exit Sub
```

## ServerCongestionGetWatch Method

The ServerCongestionGetWatch method determines if this RTclient is watching for congestion in an RTserver write buffer.

**Syntax**     *object*.ServerCongestionGetWatch (*server\_name*, *connected\_process\_name*, *watch\_status*)

where:

*object*                     is an object expression that evaluates to a TipcSrvMon object.

*server\_name*                is a string that specifies the name of the RTserver for determining watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTservers matching the MonitorScope property.

*connected\_process\_name* is a string that specifies the name of the process, connected to the RTserver, to determine the watch status for. Use "@" to indicate a watch of all processes connected to the polled RTserver.

*watch\_status*              is a Boolean value representing the watch status:

- True means watching is active.
- False means there is no watching activity.

**Return Type**     None

**C API**             TipcSrvMonServerCongestionGetWatch

**Example**           This example determines if this RTclient is watching for write buffer congestion:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ServerCongestionGetWatch("/...", "@", status)
If result = True Then
 If status = False Then
 Debug.Print "Server Congestion not being watched"
 Else
 Debug.Print "Server Congestion being watched"
 End If
Else
 Debug.Print "ServerCongestionGetWatch failed!"
End If
```

## ServerCongestionSetWatch Method

The ServerCongestionSetWatch method starts or stops watching for congestion in the RTserver write buffer of a connected process

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b> | <code>object.ServerCongestionSetWatch (server_name, connected_process_name, high_water, low_water, watch_status)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|               | <p>where:</p> <p><i>object</i> is an object expression that evaluates to a TipcSrvMon object.</p> <p><i>server_name</i> is a string that specifies the name of the RTserver for determining watch status. Wildcard names are allowed, or use "/" . . ." to indicate a watch of all RTservers matching the MonitorScope property.</p> <p><i>connected_process_name</i> is a string that specifies the name of the process, connected to the RTserver, to set the watch status for. Use "@" to indicate a watch of all processes connected to the polled RTserver.</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> is a Boolean value that sets the watch status:</p> <ul style="list-style-type: none"> <li>• True starts the watching process.</li> <li>• False stops the watching process.</li> </ul> |

**Return Type**      None

**C API**      TipcSrvMonServerCongestionSetWatch

**Example**      This example sets a watch for write buffer congestion:

```
Dim mt As TipcMt
Dim result As Boolean

mt.Lookup "mon_server_max_client_licenses_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ServerMaxClientLicensesSetWatch("/...", True)
TipcSvc1.Srv.Flush
If result = False Then
 Debug.Print "ServerMaxClientLicensesSetWatch failed!"
End If
```

## ServerConnGetWatch Method

The ServerConnGetWatch method determines if this RTclient is watching the RTserver connections.

**Syntax**     *object*.ServerConnGetWatch (*watch\_status*)  
where:  
*object*        is an object expression that evaluates to a TipcSrvMon object.  
*watch\_status* is a Boolean value representing the watch status:

- True means that watching is active.
- False means there is no watching activity.

**Return Type**     None

**C API**            TipcSrvMonServerConnGetWatch

**Example**         This example determines if this RTclient is watching RTserver connections:

```
Dim status As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

Monitor.ServerBufferGetWatch(status)
If status = False Then
 Debug.Print "Server connections not being watched"
Else
 Debug.Print "Server connections being watched"
End If
```

## ServerConnPoll Method

The ServerConnPoll method polls once for RTserver connection information.

**Syntax**     *object*.ServerConnPoll  
where *object* is an object expression that evaluates to a TipcSrvMon object.

**Return Type**     None

**C API**            TipcSrvMonServerConnPoll



**Example** This example polls for RTserver connections:

```

Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array() As String
Dim i_array() As Long
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all RTservers
Monitor.ServerConnPoll
SrvConn.Flush

'Process the poll result message
Set mt = Svc.MtLookup("mon_server_conn_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not msg is Nothing
 Debug.Print "server 1 names:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Debug.Print "server 2 names:"
 msg.NextStrArray(s_array, numItems)

 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Debug.Print "connection names:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Debug.Print "weights:"
 msg.NextInt4Array(i_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & i_array(i)
 Next i

 msg.Destroy
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend

```

## ServerConnSetWatch Method

The ServerConnSetWatch method starts or stops the watching of RTserver connections.

**Syntax**     *object*.ServerConnSetWatch (*watch\_status*)  
where:  
*object*        is an object expression that evaluates to a TipcSrvMon object.  
*watch\_status* is a Boolean value that sets the watch status:

- True starts the watching process.
- False stops the watching process.

**Return Type**     None

**C API**     TipcSrvMonServerConnSetWatch

**Example**     This example starts watching for RTserver connections. A callback is registered to handle the status message:

```
Dim mt As TipcMt
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

set mt = svc.MtLookup("mon_server_conn_status")
SrvConn.ProcessCbCreate process_cb, mt, 1, Nothing
result = Monitor.ServerConnSetWatch(True)
SrvConn.Flush
```

## ServerCpuPoll Method

The ServerCpuPoll method polls for an RTserver's CPU percentage used.

**Syntax**     *object*.ServerCpuPoll (*server\_name*)  
where:  
*object*        is an object expression that evaluates to a TipcSrvMon object.  
*server\_name* is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/" . . ." to indicate a poll of all RTservers matching the MonitorScope property.

**Return Type**     None

**C API**     TipcSrvMonServerCpuPoll

**Example**     This example polls for RTserver CPU usage:

```
Dim mt As TipcMt
Dim msg As TipcMsg

'Poll all clients including self
TipcMon1.ServerCpuPoll "/..."
TipcSvc1.Srv.Flush

'Process the poll message sent to self first
Set mt = TipcSvc1.MtLookup("mon_server_cpu_poll_call")
Set msg = TipcSvc1.Srv.MsgSearchType(10#, mt)
If msg.IsNull = False Then
 TipcSvc1.Srv.MsgProcess msg
End If

'Process the poll result message
Set mt = TipcSvc1.MtLookup("mon_server_cpu_poll_result")
Set msg = TipcSvc1.Srv.MsgSearchType(10#, mt)
While msg.IsNull = False
 On Error GoTo EndOfMessage
 eom = False
 While Not eom

 Debug.Print "Server name: " & msg.NextStr
 Debug.Print "CPU Utilization: " & msg.NextInt4

 Set msg = TipcSvc1.Srv.MsgSearchType(10#, mt)
 Wend

EndOfMessage:
 eom = True
Exit Sub

Wend
```

## ServerGeneralPoll Method

The ServerGeneralPoll method polls once for RTserver general information.

**Syntax**     *object*.ServerGeneralPoll (*server\_name*)

where:

*object*     is an object expression that evaluates to a TipcSrvMon object.

*server\_name* is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/" . . ." to indicate a poll of all RTservers matching the MonitorScope property.

**Return Type**     Boolean

**C API**    TipcSrvMonServerGeneralPoll**Example**    This example polls for RTserver general information:

```

Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array() As String
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all RTservers
Monitor.ServerGeneralPoll "/..."
SrvConn.Flush

'Process the result message
Set mt = Svc.MtLookup("mon_server_general_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 Debug.Print "RTserver: " & msg.NextStr
 Debug.Print "Ident: " & msg.NextStr
 Debug.Print "Node: " & msg.NextStr
 Debug.Print "User: " & msg.NextStr
 Debug.Print "Pid: " & msg.NextInt4
 Debug.Print "Architecture: " & msg.NextStr
 Debug.Print "Current SBRK: " & msg.NextInt4
 Debug.Print "Delta SBRK: " & msg.NextInt4
 Debug.Print "Integer format: " & msg.NextInt2
 Debug.Print "Real format: " & msg.NextInt2
 Debug.Print "cmd file name: " & msg.NextStr
 Debug.Print "no daemon flag: " & msg.NextStr
 Debug.Print "cmd mode flag: " & msg.NextStr

 Debug.Print "direct clients:"
 numItems = msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Debug.Print "direct servers:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Debug.Print "server subscribe:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Debug.Print "client subscribe:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend

```

## ServerMaxClientLicensesGetWatch Method

The ServerMaxClientLicensesGetWatch method determines if the RTserver is watching for the server cloud to exceed the licensed number of RTclient connections

**Syntax** `object.ServerMaxClientLicensesGetWatch (server_name, watch_status)`

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*server\_name* is a string that specifies the name of the RTserver for determining watch status. Wildcard names are allowed, or use "/"..." to indicate a watch of all RTservers matching the MonitorScope property.

*watch\_status* is a Boolean value representing the watch status:

- True means watching is active.
- False means there is no watching activity.

**Return Type** None

**C API** TipcSrvMonServerMaxClientLicensesGetWatch

**Example** This example determines if this RTclient is watching for client connection rejections:

```
Dim status As Boolean
Dim result As Boolean

result = TipcMon1.ServerMaxClientLicensesGetWatch("/...", status)
If result = True Then
 If status = False Then
 Debug.Print "Server Max Client Licenses not being watched"
 Else
 Debug.Print "Server Max Client Licenses being watched"
 End If
Else
 Debug.Print "ServerMaxClientLicensesGetWatch failed!"
End If
```

## ServerMaxClientLicensesSetWatch Method

The ServerMaxClientLicensesSetWatch method starts or stops watching for a server to exceed the licensed number of RTclient connections

**Syntax**     *object*.ServerMaxClientLicensesSetWatch (*server\_name*, *watch\_status*)

where:

*object*                     is an object expression that evaluates to a TipcSrvMon object.

*server\_name*                is a string that specifies the name of the RTserver for determining watch status. Wildcard names are allowed, or use "/. . ." to indicate a watch of all RTservers matching the MonitorScope property.

*watch\_status*                is a Boolean value that sets the watch status:

- True starts the watching process.
- False stops the watching process.

**Return Type**     None

**C API**     TipcSrvMonServerMaxClientLicensesSetWatch

**Example**     This example sets a watch for client connection rejections:

```
Dim mt As TipcMt
Dim result As Boolean

mt.Lookup "mon_server_congestion_status"
TipcSvc1.Srv.AddProcessCb TipcProcessCb1.Object, mt
result = TipcMon1.ServerCongestionSetWatch("/...", "@", 1, 10, True)
TipcSvc1.Srv.Flush
If result = False Then
 Debug.Print "ServerCongestionSetWatch failed!"
End If
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## ServerMsgTrafficPoll Method

The ServerMsgTrafficPoll method polls once for RTserver message traffic.

**Syntax** *object*.ServerMsgTrafficPoll (*server\_name*)

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*server\_name* is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/"... to indicate a poll of all RTservers matching the MonitorScope property.

**Return Type** None

**C API** TipcSrvMonServerMsgTrafficPoll

**Example** This example polls for RTserver message traffic information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all servers
Monitor.ServerMsgTrafficPoll "/"...
SrvConn.Flush

'Process the poll result message
Set mt = Svc.MtLookup("mon_server_msg_traffic_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 On Error Goto EndOfMessage
 eom = False
 Debug.Print "RTserver:" & msg.NextStr
 While Not eom
 Debug.Print " RTclient: " & msg.NextStr
 Debug.Print " total msgs send = " & msg.NextInt4
 Debug.Print " total msgs recv = " & msg.NextInt4
 Debug.Print " total bytes send = " & msg.NextInt4
 Debug.Print " total bytes recv = " & msg.NextInt4
 Wend
 msg.Destroy
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend
EndOfMessage:
eom = True
Exit Sub
```

## ServerNamesGetWatch Method

The ServerNamesGetWatch method determines if this RTclient is watching RTserver names.

**Syntax**     *object*.ServerNamesGetWatch (*watch\_status*)  
where:  
*object*        is an object expression that evaluates to a TipcSrvMon object.  
*watch\_status* is a Boolean value representing the watch status:

- True means watching is active.
- False means there is no watching activity.

**Return Type**     None

**C API**            TipcSrvMonServerNamesGetWatch

**Example**        This example determines if this RTclient is watching server names:

```
Dim status As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

result = Monitor.ServerNamesGetWatch(status)
If status = False Then
 Debug.Print "Server names not being watched"
Else
 Debug.Print "Server names being watched"
End If
```

## ServerNamesPoll Method

The ServerNamesPoll method polls once for RTserver names.

**Syntax**     *object*.ServerNamesPoll  
where *object* is an object expression that evaluates to a TipcSrvMon object.

**Return Type**     None

**C API**            TipcSrvMonServerNamesPoll



**Example** This example polls for RTserver names:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array() As String
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll all servers
Monitor.ServerNamesPoll
SrvConn.Flush

'Process the poll result message
Set mt = Svc.MtLookup("mon_server_names_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 Debug.Print "RTserver names:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 Debug.Print "RTserver info strings:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i

 msg.Destroy
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend
```

## ServerNamesSetWatch Method

The ServerNamesSetWatch method starts or stops the watching of RTserver names.

**Syntax** *object*.ServerNamesSetWatch (*watch\_status*)

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*watch\_status* is a Boolean value that sets the watch status:

- True starts the watching process.
- False stops the watching process.

**Return Type** None

**C API**      TipcSrvMonServerNamesSetWatch

**Example**      This example starts watching for RTserver names. A callback is registered to handle the status message:

```
Dim mt As TipcMt
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

set mt = svc.MtLookup("mon_server_names_status"
SrvConn.ProcessCbCreate process_cb, mt, 1, Nothing
Monitor.ServerNamesSetWatch(True)
SrvConn.Flush
```

**ServerOptionPoll Method**

---

The ServerOptionPoll method polls once for RTserver options.

**Syntax**      *object*.ServerOptionPoll (*server\_name*, *option\_name*)

where:

*object*              is an object expression that evaluates to a TipcSrvMon object.

*server\_name*      is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/"... to indicate a poll of all RTservers matching the MonitorScope property.

*option\_name*      is a string that specifies the name of the option to poll for information. Use "@" to indicate a poll of all options.

**Return Type**      None

**C API**      TipcSrvMonServerOptionPoll

**Example**      This example polls for RTserver option information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean
Dim numItems As Long
Dim i As Long
Dim s_array() As String
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

```
'Poll all servers
Monitor.ServerOptionPoll "/...", "@"
SrvConn.Flush

'Process the poll result message
Set mt = Svc.MtLookup("mon_server_option_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 On Error Goto EndOfMessage
 eom = False
 Debug.Print "RTserver:" & msg.NextStr
 While Not eom
 Debug.Print " option: " & msg.NextStr
 Debug.Print " type: " & msg.NextInt2
 Debug.Print " value: " & msg.NextStr
 Debug.Print " required: " & msg.NextInt4
 Debug.Print " legal values:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i
 Wend
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend

EndOfMessage:
 eom = True
Exit Sub
```

## ServerRoutePoll Method

The ServerRoutePoll method polls once for RTserver routing information.

|                         |                                                                                                                                                                                     |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>           | <i>object</i> .ServerRoutePoll ( <i>server_name</i> , <i>dest_server_name</i> )                                                                                                     |
|                         | where:                                                                                                                                                                              |
| <i>object</i>           | is an object expression that evaluates to a TipcSrvMon object.                                                                                                                      |
| <i>server_name</i>      | is a string that specifies the name of the RTserver to poll.<br>Wildcard names are allowed, or use "/. . ." to indicate a poll of all RTservers matching the MonitorScope property. |
| <i>dest_server_name</i> | is a string that specifies the name of the destination RTserver whose route should be polled. Use to indicate a poll of all options.                                                |
| <b>Return Type</b>      | None                                                                                                                                                                                |
| <b>C API</b>            | TipcSrvMonServerRoutePoll                                                                                                                                                           |

**Example** This example polls for RTserver routing information:

```
Dim mt As TipcMt
Dim msg As TipcMsg
Dim eom As Boolean
Dim numItems As Long
Dim i As Long
Dim s_array() As String

'Poll all servers
Monitor.ServerRoutePoll "/...", "@"
SrvConn.Flush

'Process the poll result message
Set mt = Svc.MtLookup("mon_server_route_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 On Error Goto EndOfMessage
 eom = False
 Debug.Print "RTserver:" & msg.NextStr
 While Not eom
 Debug.Print " destination RTserver: " & msg.NextStr
 Debug.Print " connected servers:"
 msg.NextStrArray(s_array(), numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i
 Debug.Print " distance: " & msg.NextInt4
 Debug.Print " route to server:"
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i
 Wend
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend

EndOfMessage:
eom = True
Exit Sub
```

## ServerTimePoll Method

The ServerTimePoll method polls once for RTserver time information.

**Syntax** *object*.ServerTimePoll (*server\_name*)

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*server\_name* is a string that specifies the name of the RTserver to poll. Wildcard names are allowed, or use "/" . . ." to indicate a poll of all RTservers matching the MonitorScope property.

**Return Type** None

**C API**    TipcSrvMonServerTimePoll**Example**    This example polls for time information from each RTserver:

```

Dim mt As TipcMt
Dim msg As TipcMsg

'Poll all servers
Monitor.ServerTimePoll "/"..."
SrvConn.Flush

'Process the poll result message
Set mt = Svc.MtLookup("mon_server_time_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 Debug.Print "RTserver: " & msg.NextStr
 Debug.Print " wall time: " & msg.NextReal8
 Debug.Print " wall time: " & msg.NextStr
 Debug.Print " CPU time: " & msg.NextReal8
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend

```

## SubjectNamesGetWatch Method

---

The SubjectNamesGetWatch method determines if this RTclient is watching subject names.

**Syntax**    *object*.SubjectNamesGetWatch (*watch\_status*)

where:

*object*        is an object expression that evaluates to a TipcSrvMon object.*watch\_status* is a Boolean value representing the watch status:

- True means watching is active.
- False means there is no watching activity.

**Return Type**    None**C API**    TipcSrvMonSubjectNamesGetWatch**Example**    This example determines if this RTclient is watching subject names:

```

Dim status As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

result = Monitor.SubjectNamesGetWatch(status)
If status = False Then

```

```

 Debug.Print "Subject names not being watched"
 Else
 Debug.Print "Subject names being watched"
 End If

```

## SubjectNamesPoll Method

---

The SubjectNamesPoll method polls once for subject names.

**Syntax**     *object*.SubjectNamesPoll

where *object* is an object expression that evaluates to a TipcSrvMon object.

**Return Type**     None

**C API**     TipcSrvMonSubjectNamesPoll

**Example**     This example polls for all subject names:

```

Dim mt As TipcMt
Dim msg As TipcMsg
Dim numItems As Long
Dim i As Long
Dim s_array() As String
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

'Poll RTserver for subject names
Monitor.SubjectNamesPoll
SrvConn.Flush

'Process the poll result message
Set mt = Svc.MtLookup("mon_subject_names_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 Debug.Print " Subject names:" & msg.NextStr
 msg.NextStrArray(s_array(), numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend

```

## SubjectNamesSetWatch Method

---

The SubjectNamesSetWatch method starts or stops the watching of subject names.

**Syntax** *object*.SubjectNamesSetWatch (*watch\_status*)

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*watch\_status* is a Boolean value that sets the watch status:

- True starts the watching process.
- False stops the watching process.

**Return Type** None

**C API** TipcSrvMonSubjectNamesSetWatch

**Example** This example starts watching for subjects names. A callback is registered to handle the status message:

```
Dim mt As TipcMt
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

set mt = svc.MtLookup("mon_subject_names_status"
SrvConn.ProcessCbCreate process_cb, mt, 1, Nothing
Monitor.SubjectNamesSetWatch(True)
SrvConn.Flush
```

## SubscribeGetWatch Method

---

The SubscribeGetWatch method determines if this RTclient is watching for RTclients that are subscribing to a subject.

**Syntax** *object*.SubscribeGetWatch (*subject\_name*, *watch\_status*)

where:

*object* is an object expression that evaluates to a TipcSrvMon object.

*subject\_name* is a string that specifies the name of the subject for determining watch status. Wildcard names are allowed, or use "/"... to indicate a watch of all subjects matching the MonitorScope property.

*watch\_status* is a Boolean value representing the watch status:

- True means that watching is active.
- False means there is no watching activity.

**Return Type**      None

**C API**            TipcSrvMonSubjectSubscribeGetWatch

**Example**        This example determines if this RTclient is watching clients that are subscribed to a subject:

```
Dim status As Boolean
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

Monitor.SubscribeGetWatch("/new", status)
If status = False Then
 Debug.Print "/new subject not being watched"
Else
 Debug.Print "/new subject being watched"
End If
```

## SubscribePoll Method

The SubscribePoll method polls once for the clients that are subscribing to a subject.

**Syntax**        *object.SubscribePoll (subject\_name)*  
                  where:  
*object*            is an object expression that evaluates to a TipcSrvMon object.  
*subject\_name*    is a string that specifies the name of the subject to poll for subject subscription information. Wildcard names are allowed, or use "/. . ." to indicate a poll of all subjects matching the MonitorScope property.

**Return Type**      None

**C API**            TipcSrvMonSubjectSubscribePoll



**Example** This example polls for all clients that are subscribing to "/db":

```
Dim mt As TipcMt, msg As TipcMsg, i As Long
Dim numItems As Long, eom As Boolean
Dim s_array() As String

'Poll "/db" subject for all clients
Monitor.SubscribePoll "/db"
SrvConn.Flush

'Process the poll result message
Set mt = Svc.MtLookup("mon_subscribe_poll_result")
Set msg = SrvConn.MsgSearchType(10.0, mt)
While Not Msg is Nothing
 On Error Goto EndOfMessage
 eom = False
 While Not eom
 Debug.Print "Subject:" & msg.NextStr
 msg.NextStrArray(s_array, numItems)
 For i = 0 To numItems - 1
 Debug.Print " " & s_array(i)
 Next i
 Wend
 Set msg = SrvConn.MsgSearchType(10.0, mt)
Wend

EndOfMessage:
 eom = True
```

## SubscribeSetWatch Method

The SubscribeSetWatch method starts or stops the watching of clients that are subscribing to a subject.

**Syntax** *object.SubscribeSetWatch (subject\_name, watch\_status)*

where:

- object* is an object expression that evaluates to a TipcSrvMon object.
- client\_name* is a string that specifies the name of the subject for setting watch status. Wildcard names are allowed, or use "/"... to indicate a watch of all subjects matching the MonitorScope property.
- watch\_status* is a Boolean value that sets the watch status:
  - True starts the watching process.
  - False stops the watching process.

**Return Type** None

**C API** TipcSrvMonSubjectSubscribeSetWatch

**Example** This example starts watching for clients that are subscribed to "/db". A callback is registered to handle the status message:

```
Dim mt As TipcMt
Monitor as New TipcSrvMon
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
Monitor.Init SrvConn

set mt = svc.MtLookup("mon_subject_subscribe_status")
SrvConn.ProcessCbCreate process_cb, mt, 1, Nothing
result = Monitor.SubscribeSetWatch("/db", True)
SrvConn.Flush
```

## TipcSrvMonExt Class

TipcSrvMonExt — class for defining, setting, or deleting fields for RTclient extension data.

**Description** The TipcSrvMonExt class is used to define, set, or delete fields in an RTclient MON\_CLIENT\_EXT\_POLL\_RESULT message. A MON\_CLIENT\_EXT\_POLL\_RESULT message contains the extension data returned to the RTclient that issues a TipcSrvMon class ClientExtPoll method.

**Methods**

- Create Method
- Destroy Method
- SetBinary Method
- SetBool Method
- SetBoolArray Method
- SetInt2 Method
- SetInt2Array Method
- SetInt4 Method
- SetInt4Array Method
- SetInt8 Method
- SetInt8Array Method
- Delete Method
- SetReal4 Method
- SetReal4Array Method
- SetReal8 Method
- SetReal8Array Method
- SetReal16 Method
- SetReal16Array Method
- SetStr Method
- SetStrArray Method
- SetUtf8 Method
- SetUtf8Array Method

**Properties and Events** None

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## TipcSrvMonExt Class Methods

### Create Method

The Create method creates the TipcSrvMonExt object and assigns the connection it will use to publish monitoring messages.

**Syntax**     *object*.Create *SrvConn*  
where:  
*object*     is an object expression that evaluates to a TipcSrvMonExt object.  
*SrvConn*    is an object expression that evaluates to a TipcSrvConn object.

**Return Type**     None

**C API**     No comparable C API function

**Example**     This example creates the TipcSrvMon object:

```
MonitorExt as New TipcSrvMonExt
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
MonitorExt.Create SrvConn
```

### Delete Method

The Delete method 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 the Delete method.

**Syntax**     *object*.Delete (*field\_name*)  
where:  
*object*     is an object expression that evaluates to a TipcSrvMonExt object.  
*field\_name*    is name of the field to remove from the RTclient's  
                 MON\_CLIENT\_EXT\_POLL\_RESULT message.

|             |                        |
|-------------|------------------------|
| Return Type | None                   |
| C API       | TipcSrvMonClientDelete |

## Destroy Method

The Destroy method destroys the TipcSrvMon object.

|             |                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <div><i>object</i>.Destroy</div> <div>where <i>object</i> is an object expression that evaluates to a TipcSrvMon object.</div> |
| Return Type | None                                                                                                                           |
| C API       | No comparable C API function                                                                                                   |
| Example     | This example destroys a monitor object:                                                                                        |

```
MonitorExt as New TipcSrvMonExt
Dim svc as New TipcSvc
Dim SrvConn as TipcSrvConn
Set SrvConn = svc.TipcSrvConnCreate("", "", "", "")
SrvConn.Open
MonitorExt.Create SrvConn
MonitorExt.Destroy
```

## SetBinary Method

The SetBinary method adds or updates a field of type BINARY in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

|        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax | <div><i>object</i>.SetBinary (<i>field_name</i>, <i>data</i>, <i>size</i>)</div> <div>where:</div> <div><i>object</i> is an object expression that evaluates to a TipcSrvMonExt object.</div> <div><i>field_name</i> is the name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.</div> <div><i>data</i> is the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message.</div> <div><i>size</i> is the number of bytes in the data.</div> |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

|             |                                                       |
|-------------|-------------------------------------------------------|
| Return Type | None                                                  |
| C API       | TipcSrvMonExtSetBinary                                |
| Example     | See the example for ClientExtPoll Method on page 473. |

## SetBool Method

The SetBool method adds or updates a field of type BOOL in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**    *object*.SetBool (*field\_name*, *data*)

where:

*object*        is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name*    is the name of a field in the RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data*          is the RTclient information to write to *field\_name* in the MON\_CLIENT\_EXT\_POLL\_RESULT message.

|             |                                                           |
|-------------|-----------------------------------------------------------|
| Return Type | None                                                      |
| C API       | TipcSrvMonExtSetBool                                      |
| Example     | See the example for the ClientExtPoll Method on page 473. |

## SetBoolArray Method

The SetBoolArray method adds or updates a field containing an array of type BOOL in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**    *object*.SetBoolArray (*field\_name*, *data*, *size*)

where:

*object*        is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name*    is the name of a field in the RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data* is the RTclient information to write to *field\_name* in the MON\_CLIENT\_EXT\_POLL\_RESULT message.

*size* is the number of elements in the data.

- Return Type** None
- C API** TipcSrvMonExtSetBoolArray
- Example** See the example for the ClientExtPoll Method on page 473.

## SetInt2 Method

---

The SetInt2 method adds or updates a field a field of type INT2 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax** *object*.SetInt2 (*field\_name*, *data*)

where:

*object* is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name* is the name of a field in the RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data* is the RTclient information to write to *field\_name* in the MON\_CLIENT\_EXT\_POLL\_RESULT message.

- Return Type** None
- C API** TipcSrvMonExtSetInt2
- Example** See the example for the ClientExtPoll Method on page 473.

## SetInt2Array Method

---

The SetInt2Array method adds or updates a field containing an array of type INT2 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**     *object*.SetInt2Array (*field\_name*, *data*, *size*)

where:  
*object*        is an object expression that evaluates to a TipcSrvMonExt object.  
*field\_name*    is the name of a field in the RTclient's  
                 MON\_CLIENT\_EXT\_POLL\_RESULT message.  
*data*           is the RTclient information to write to *field\_name* in the  
                 MON\_CLIENT\_EXT\_POLL\_RESULT message.  
*size*           is the number of elements in the data.

**Return Type**     None

**C API**            TipcSrvMonExtSetInt2Array

**Example**        See the example for the ClientExtPoll Method on page 473.

## SetInt4 Method

---

The SetInt4 method adds or updates a field a field of type INT4 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**     *object*.SetInt4 (*field\_name*, *data*)

where:  
*object*        is an object expression that evaluates to a TipcSrvMonExt object.  
*field\_name*    is the name of a field in the RTclient's  
                 MON\_CLIENT\_EXT\_POLL\_RESULT message.  
*data*           is the RTclient information to write to *field\_name* in the  
                 MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Return Type**     None



**C API**     TipcSrvMonExtSetInt4

**Example**     See the example for the ClientExtPoll Method on page 473.

## SetInt4Array Method

---

The SetInt4Array method adds or updates a field containing an array of type INT4 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**     *object*.SetInt4Array (*field\_name*, *data*, *size*)

where:

*object*        is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name*    is the name of a field in the RTclient's  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data*          is the RTclient information to write to *field\_name* in the  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

*size*          is the number of elements in the data.

**Return Type**     None

**C API**     TipcSrvMonExtSetInt4Array

**Example**     See the example for the ClientExtPoll Method on page 473.

## SetInt8 Method

---

The SetInt8 method adds or updates a field a field of type INT8 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**     *object*.SetInt8 (*field\_name*, *data*)

where:

*object*        is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name*    is the name of a field in the RTclient's  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data* is the RTclient information to write to *field\_name* in the MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Return Type**      None

**C API**      TipcSrvMonExtSetInt8

**Example**      See the example for the ClientExtPoll Method on page 473.

## SetInt8Array Method

---

The SetInt8Array method adds or updates a field containing an array of type INT8 in an RTclient’s MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**      *object*.SetInt8Array (*field\_name*, *data*, *size*)

- where:
- object* is an object expression that evaluates to a TipcSrvMonExt object.
  - field\_name* is the name of a field in the RTclient’s MON\_CLIENT\_EXT\_POLL\_RESULT message.
  - data* is the RTclient information to write to *field\_name* in the MON\_CLIENT\_EXT\_POLL\_RESULT message.
  - size* is the number of elements in the data.

**Return Type**      None

**C API**      TipcSrvMonExtSetInt8Array

**Example**      See the example for the ClientExtPoll Method on page 473.

## SetReal4 Method

---

The SetReal4 method adds or updates a field a field of type REAL4 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**     *object.SetReal4 (field\_name, data)*

where:

*object*     is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name* is the name of a field in the RTclient's  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data*     is the RTclient information to write to *field\_name* in the  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Return Type**     None

**C API**     TipcSrvMonExtSetReal4

**Example**     See the example for the ClientExtPoll Method on page 473.

## SetReal4Array Method

---

The SetReal4Array method adds or updates a field containing an array of type REAL4 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**     *object.SetReal4Array field\_name, data, size)*

where:

*object*     is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name* is the name of a field in the RTclient's  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data*     is the RTclient information to write to *field\_name* in the  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

*size*     is the number of elements in the data.

**Return Type**     None

**C API**     TipcSrvMonExtSetReal4Array

**Example** See the example for the ClientExtPoll Method on page 473.

## SetReal8 Method

---

The SetReal8 method adds or updates a field a field of type REAL8 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax** `object.SetReal8 (field_name, data)`

where:

*object* is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name* is the name of a field in the RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data* is the RTclient information to write to *field\_name* in the MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Return Type** None

**C API** TipcSrvMonExtSetReal8

**Example** See the example for the ClientExtPoll Method on page 473.

## SetReal8Array Method

---

The SetReal8Array method adds or updates a field containing an array of type REAL8 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax** `object.SetReal8Array (field_name, data, size)`

where:

*object* is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name* is the name of a field in the RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data* is the RTclient information to write to *field\_name* in the MON\_CLIENT\_EXT\_POLL\_RESULT message.

*size* is the number of elements in the data.

|                    |                                                           |
|--------------------|-----------------------------------------------------------|
| <b>Return Type</b> | None                                                      |
| <b>C API</b>       | TipcSrvMonExtSetReal8Array                                |
| <b>Example</b>     | See the example for the ClientExtPoll Method on page 473. |

## SetReal16 Method

The SetReal16 method adds or updates a field a field of type REAL16 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

|                   |                                                                                                      |
|-------------------|------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>     | <code>object.SetReal16 (field_name, data)</code>                                                     |
|                   | where:                                                                                               |
| <i>object</i>     | is an object expression that evaluates to a TipcSrvMonExt object.                                    |
| <i>field_name</i> | is the name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message.                         |
| <i>data</i>       | is the RTclient information to write to <i>field_name</i> in the MON_CLIENT_EXT_POLL_RESULT message. |

|                    |                                                           |
|--------------------|-----------------------------------------------------------|
| <b>Return Type</b> | None                                                      |
| <b>C API</b>       | TipcSrvMonExtSetReal16                                    |
| <b>Example</b>     | See the example for the ClientExtPoll Method on page 473. |

## SetReal16Array Method

The SetReal16Array method adds or updates a field containing an array of type REAL16 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

|                   |                                                                              |
|-------------------|------------------------------------------------------------------------------|
| <b>Syntax</b>     | <code>object.SetReal16Array (field_name, data, size)</code>                  |
|                   | where:                                                                       |
| <i>object</i>     | is an object expression that evaluates to a TipcSrvMonExt object.            |
| <i>field_name</i> | is the name of a field in the RTclient's MON_CLIENT_EXT_POLL_RESULT message. |

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

*data* is the RTclient information to write to *field\_name* in the MON\_CLIENT\_EXT\_POLL\_RESULT message.

*size* is the number of elements in the data.

**Return Type** None

**C API** TipcSrvMonExtSetReal16Array

**Example** See the example for the ClientExtPoll Method on page 473.

## SetStr Method

---

The SetStr method adds or updates a field a field of type STR in an RTclient’s MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax** *object*.SetStr (*field\_name*, *data*)

where:

*object* is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name* is the name of a field in the RTclient’s MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data* is the RTclient information to write to *field\_name* in the MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Return Type** None

**C API** TipcSrvMonExtSetStr

**Example** See the example for the ClientExtPoll Method on page 473.

## SetStrArray Method

---

The SetStrArray method adds or updates a field containing an array of type STR in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**     *object.SetStrArray (field\_name, data, size)*

where:

*object*     is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name* is the name of a field in the RTclient's  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data*     is the RTclient information to write to *field\_name* in the  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

*size*     is the number of elements in the data.

**Return Type**     None

**C API**     TipcSrvMonExtSetStrArray

**Example**     See the example for the ClientExtPoll Method on page 473.

## SetUtf8 Method

---

The SetUtf8 method adds or updates a field a field of type UTF8 in an RTclient's MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**     *object.SetUtf8 (field\_name, data)*

where:

*object*     is an object expression that evaluates to a TipcSrvMonExt object.

*field\_name* is the name of a field in the RTclient's  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

*data*     is the RTclient information to write to *field\_name* in the  
MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Return Type**     None

**C API**      TipcSrvMonExtSetUtf8

**Example**      See the example for the ClientExtPoll Method on page 473.

## SetUtf8Array Method

---

The SetUtf8Array method adds or updates a field containing an array of type UTF8 in an RTclient’s MON\_CLIENT\_EXT\_POLL\_RESULT message.

**Syntax**      *object*.SetUtf8Array (*field\_name*, *data*, *size*)

- where:
- object*      is an object expression that evaluates to a TipcSrvMonExt object.
  - field\_name*      is the name of a field in the RTclient’s MON\_CLIENT\_EXT\_POLL\_RESULT message.
  - data*      is the RTclient information to write to *field\_name* in the MON\_CLIENT\_EXT\_POLL\_RESULT message.
  - size*      is the number of elements in the data.

**Return Type**      None

**C API**      TipcSrvMonExtSetUtf8Array

**Example**      See the example for the ClientExtPoll Method on page 473.

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED



## TipcSubjects Class

---

TipcSubjects — user class for managing publish-subscribe participation

|                    |                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | The TipcSubjects class provides an interface to the subjects to which an RTclient is subscribed. It allows you to add, remove, retrieve and count the subjects contained within. For more information on subjects, see the <i>TIBCO SmartSockets User's Guide</i> . |
| <b>Methods</b>     | <ul style="list-style-type: none"><li>• Add Method</li><li>• Remove Method</li><li>• RemoveAll Method</li></ul>                                                                                                                                                     |
| <b>Properties</b>  | <ul style="list-style-type: none"><li>• Count Property</li><li>• Item Property</li></ul>                                                                                                                                                                            |
| <b>Events</b>      | None                                                                                                                                                                                                                                                                |

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## TipcSubjects Class Methods

### Add Method

The Add method subscribes to and determines participation in the load balancing calculations for the specified subject.

**Syntax**     *object*.Add *subject*, *lb\_status*

where:

*object*        is an object expression that evaluates to a TipcSubjects object.

*subject*        is a string that contains the publish-subscribe subject to subscribe to.

*lb\_status*      is a Boolean value that specifies whether to participate in load balancing for this subject:

- `True` means the RTclient receives messages using the load balancing mode set for this subject.
- `False` means that the RTclient always receives messages published to this subject.

**Return Type**     None

**C API**            TipcSrvConnSubjectSetSubscribeLb

**Example**        This example subscribes to the two subjects, one as part of the load balancing group, and one to always receive all messages:

```
SrvConn.Subjects.Add "/demo/sub_one",False
SrvConn.Subjects.Add "/demo/load_bal",True
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## Remove Method

---

The Remove method unsubscribes an RTclient from the specified subject.

**Syntax** `object.Remove subject`

where:

*object* is an object expression that evaluates to a TipcSubjects object.

*subject* is a string that contains the publish-subscribe subject to stop subscribing to.

**Return Type** None

**C API** TipcSrvConnSubjectSetSubscribeLb

**Example** This example unsubscribes from a subject:  
`SrvConn.Subjects.Remove "/demo/sub_one"`

## RemoveAll Method

---

The RemoveAll method removes all subjects that a client is subscribed to.

**Syntax** `object.RemoveAll`

where *object* is an object expression that evaluates to a TipcSubjects object.

**Return Type** None

**C API** No comparable C API function

**Example** This example removes all subjects a client is subscribed to:  
`SrvConn.Subjects.RemoveAll`

## TipcSubjects Class Properties

---

### Count Property

---

The Count property returns the number of subjects contained in the TipcSubjects object. This is a read-only value.

**Syntax**     *object*.Count  
where *object* is an object expression that evaluates to a TipcSubjects object.

**Data Type**     Long (Integer)

**C API**     No direct counterpart in the C API, similar function is achieved within the Monitoring API.

**Example**     This example prints out the number of subjects to which this RTclient is subscribed:

```
Dim ct As Long
ct = Svc.Subject.Count
Debug.Print "subscribed to " & Format(ct) & " subjects"
```

### Item Property

---

The Item property returns the subject name being subscribed to with the specified index. The first subject is specified by index 0. This property is read-only and can only be modified by the Add and Remove methods.

**Syntax**     *object*.Item(*index*)  
where:  
*object*     is an object expression that evaluates to a TipcSubjects object.  
*index*     is a long integer that specifies which subscribed subject to return. Using 0 returns the first field.

**Data Type**     String

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

**Remarks** The word "item" has special meaning in most COM environments; the `Item` keyword can be omitted and the index in parentheses applied directly to the object. See the example below.

**C API** No comparable C API function

**Example** This example uses the `Item` property, first directly and then indirectly, to display the first two subjects to which this `RTclient` is subscribed:

```
Debug.Print SrvConn.Subjects.Item(0)
Debug.Print SrvConn.Subjects(1)
```

This example prints out all of the subjects subscribed to:

```
Dim ct As Long
Dim i As Long

ct = SrvConn.Subjects.Count
For i = 0 to ct - 1
 Debug.Print SrvConn.Subjects(i)
Next i
```

# TipcSvc Class

TipcSvc — user class that provides an interface for creating objects and enables options to be set

**Description** The TipcSvc class provides an interface to the options used to connect with RTserver and SmartSockets messages. It allows you to set various timeouts, to specify to which RTservers to connect, and to create new SmartSockets messages. For more information on RTserver, see the *TIBCO SmartSockets User's Guide*.

## Methods

- CreateOption Method
- GetOption Method
- MtCreate Method
- MtLookupByNum Method
- ParseStr Method
- SetOption Method
- TipcDispatcherCreateDetached Method
- TipcSrvConnCreateNamed Method
- XmlCreate Method
- GetEnv Method
- MsgCreate Method
- MtLookup Method
- ParseFile Method
- PutEnv Method
- TipcDispatcherCreate Method
- TipcSrvConnCreate Method
- TMsgFileCreate Method

## Properties

- DefaultMsgPriority Property
- EnableControlMsgs Property
- LastErrorStr Property
- ServerAutoFlushSize Property
- ServerKeepAliveTimeout Property
- ServerReadTimeout Property
- ServerStartMaxTries Property
- ServerWriteTimeout Property
- DefaultSubjectPrefix Property
- LastErrorNum Property
- MonitorScope Property
- ServerDeliveryTimeout Property
- ServerMsgSend Property
- ServerStartDelay Property
- ServerStartTimeout Property

**Events** None

## TipcSvc Class Methods

---

### CreateOption Method

---

The CreateOption method creates an option and sets its value.

**Syntax** `object.createOption option_name, option_val, option_type`  
 where:  
*object* is an object expression that evaluates to a TipcSvc object.  
*option\_name* is the name of the option to be created.  
*option\_val* is the initial value of the option, and depends on *option\_type*.  
*option\_type* is the type of the option. Valid values are:

- `tipcOptionBoolean` creates a boolean option.
- `tipcOptionEnum` creates an enumerated option.
- `tipcOptionListEnum` creates an enumerated list option.
- `tipcOptionListNumeric` creates a numeric list option.
- `tipcOptionListString` creates a string list option.
- `tipcOptionNumeric` creates a numeric option.
- `tipcOptionString` creates a string option.

**Return Type** None

**C API** TutOptionCreate and TutOptionSet\*.

**Example** This example creates a new boolean option:

```
Dim svc as New TipcSvc
svc.CreateOption "bool_opt", true, tipcOptionBoolean
```

### GetEnv Method

---

The GetEnv method gets a value from the environment.

**Syntax** `object.GetEnv name`

where:  
*object* is an object expression that evaluates to a TipcSvc object.  
*name* is the name of environment variable whose value is to be retrieved.

|             |                                                                                                                                                                                                                                                                                                |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Return Type | A string object representing the environment variable.                                                                                                                                                                                                                                         |
| Remarks     | <p>The environment is a list of name / value pairs that is set by the operating system for each process.</p> <p>On Windows, GetEnv gets the value from the runtime associated with the SmartSockets libraries, which may be different that the runtime that the application is running in.</p> |
| C API       | TutGetenv                                                                                                                                                                                                                                                                                      |
| Example     | <p>This example gets the value of RTHOME from the environment:</p> <pre>Dim envVar As String Set envVar = Svc.GetEnv("RTHOME")</pre>                                                                                                                                                           |

## GetOption Method

The GetOption method retrieves the value of a SmartSockets option.

**Syntax** *object*.GetOption *option\_name*, *option\_val*  
where:  
*object* is an object expression that evaluates to a TipcSvc object.  
*option\_name* is a string specifying the name of the option.  
*option\_val* is a variant returning the option value.

|             |                                  |
|-------------|----------------------------------|
| Return Type | None                             |
| C API       | TutOptionLookup, TutOptionGetXXX |



**Example** This example looks up the `unique_subject` option:

```
Dim svc as New TipcSvc
Dim val as Variant
Dim unique_subject as String
Svc.GetOption("unique_subject", val)
Unique_subject = val
Debug.Print unique_subject
```

## MsgCreate Method

The `MsgCreate` method is similar to the `TipcMsg.Create` method. It creates a message object containing a SmartSockets message of the specified type.

**Syntax** `object.MsgCreate mt`

where:

`object` is an object expression that evaluates to a `TipcSvc` object.

`mt` is an initialized `TipcMt` object.

**Return Type** `TipcMsg`

**Remarks** It is often easier to use this method to create new messages than to call the lower-level methods of `TipcMsg` directly. For more information, see the *TIBCO SmartSockets User's Guide*.

**C API** `TipcMsgCreate`

**Example** This example puts a new info message in the `msg` object:

```
Dim mt As TipcMt, msg as TipcMsg

set mt = svc.MtLookup(("info"))
Set msg = Svc.MsgCreate(mt)
```

## MtCreate Method

The MtCreate method creates a new message type object containing a SmartSockets message of the specified type.

**Syntax**     *object*.MtCreate (*name*, *num*, *grammar*)

where:

*object*     is an object expression that evaluates to a TipcSvc object.

*name*       is a string representing the name of the message type. It is not case-sensitive.

*num*        is a long integer that specifies the numerical ID being assigned to the new message type.

*grammar*   is a string that specifies the description of the message's format used for reading and writing message files.

**Return Type**     TipcMt

**Remarks**       See the *TIBCO SmartSockets User's Guide* for more information about grammar and field types, as well as numeric message IDs. It is often easier to use this method to create new message types than to call the lower-level methods of TipcMt directly.

**C API**            TipcMtCreate

**Example**        This example creates a new message type *two\_strings* with numeric ID 2003, and a grammar of two string fields:

```
Dim mt as TipcMt
Set mt = Svc.MtCreate("two_strings", 2003, "str str")
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## MtLookup Method

---

The MtLookup method creates a new message type object containing a SmartSockets message of the specified pre-existing type, indicated by the descriptive name provided.

**Syntax**     *object*.MtLookup (*name*)  
where:  
*object*    is an object expression that evaluates to a TipcSvc object.  
*name*     is a string representing the name of the message type. It is not case-sensitive.

**Return Type**     TipcMt

**C API**            TipcMtLookup

**Example**        This example shows two different ways to initialize a TipcMt object to the numeric\_data message type:

```
Dim mt as TipcMt
Dim svc as New TipcSvc
Set mt = Svc.MtLookup("numeric_data")
```

## MtLookupByNum Method

---

The MtLookup method creates a new message type object containing a SmartSockets message of the specified pre-existing type, indicated by the numeric ID provided.

**Syntax**     *object*.MtLookupByNum(*num*)  
where:  
*object*    is an object expression that evaluates to a TipcSvc object.  
*num*     is a long integer representing the assigned numeric ID of the message type.

**Return Type**     TipcMt

**C API**            TipcMtLookup

**Example** This example shows two different ways to initialize a TipcMt object to the numeric\_data message type:

```
Dim mt as TipcMt
Set mt = Svc.MtLookupByNum(-22)
```

## ParseFile Method

The ParseFile method parses a SmartSockets command file. ParseFile returns True if the file was accessible, False if it was unable to access the file.

**Syntax** *object*.ParseFile (*filename*)  
where:  
*object* is an object expression that evaluates to a TipcSvc object.  
*filename* is a string that specifies the name of the command file to parse, including the drive and pathname if needed.

**Return Type** Boolean, where:  
True means the file was accessible.  
False means the file could not be accessed.

**Remarks** A True result does not necessarily imply the operations specified in the file were completed successfully. For more information on command files, see the *TIBCO SmartSockets User's Guide*.

**C API** TutCommandParseFile

**Example** This example illustrates how to use ParseFile to read in a SmartSockets command file located on the C drive:

```
Dim success as Boolean
success = Svc.ParseFile("c:\demos\init.cm")
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## ParseStr Method

The ParseStr method parses a SmartSockets command string.

**Syntax**     `object.ParseStr command_string`  
where:  
`object`            is an object expression that evaluates to a TipcSvc object.  
`command_string` is a string that specifies the SmartSockets command to parse.

**Return Type**     None

**Remarks**     For more information on valid commands, see the *TIBCO SmartSockets User's Guide*.

**C API**     TutCommandParseStr

**Example**     This example uses the ParseStr method to set the RTclient's project:  
`Svc.ParseStr("setopt project activex_demo")`

## PutEnv Method

The PutEnv method sets a SmartSockets environment variable.

**Syntax**     `object.PutEnv name_and_value`  
where:  
`object`            is an object expression that evaluates to a TipcSvc object.  
`name_and_value` is both the name and value to be put into the environment, separated by an equal sign (=).

**Return Type**     None

**Remarks**     The environment is a list of name/value pairs that is set by the operating system for each process.  
  
On Windows, PutEnv gets the value from the runtime associated with the SmartSockets libraries, which may be different that the runtime that the application is running in.

**C API**     TutPutenv

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

**Example** This example sets the value of RTHOME in the environment:

```
Svc.PutEnv("YOUR_ENV_VAR=something")
```

## SetOption Method

---

The SetOption method sets a SmartSockets option.

**Syntax** *object.SetOption option\_name, option\_val*  
where:  
*object* is an object expression that evaluates to a TipcSvc object.  
*option\_name* is a string specifying the name of the option to set.  
*option\_val* is a Variant specifying the new value of the option.

**Return Type** None

**C API** TutOptionLookup, TutOptionSetXXX

**Example** This example sets the unique\_subject option:

```
Svc.SetOption "Unique_Subject", "/foobar"
```

## TipcDispatcherCreate Method

---

The TipcDispatcherCreate method creates a dispatcher object.

**Syntax** *object.TipcDispatcherCreate*  
where *object* is an object expression that evaluates to a TipcSvc object.

**Return Type** TipcDispatcher

**C API** TipcDispatcherCreate

**Example** This example creates a TipcDispatcher object:

```
Dim dispatcher as TipcDispatcher
Dispatcher = Svc.TipcDispatcherCreate
```

## TipcDispatcherCreateDetached Method

The TipcDispatcherCreateDetached method creates a dispatcher object that will run in its own thread.

**Syntax** `object.TipcDispatcherCreateDetached`  
 where *object* is an object expression that evaluates to a TipcSvc object.

**Return Type** TipcDispatcher

**C API** TipcDispatcherCreate

**Example** This example creates a TipcDispatcher object:

```
Dim dispatcher as TipcDispatcher
Dispatcher = Svc.TipcDispatcherCreateDetached
```

## TipcSrvConnCreate Method

The TipcSrvConnCreate method 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. The TipcSrvConnCreate method creates a TipcSrvConn object.

**Syntax** `object.TipcSrvConnCreate unique_subject, project, server_names, def_subject_prefix`  
 where:  
*object* is an object expression that evaluates to a TipcSvc object.  
*unique\_subject* is a unique subject to use for the connection, or "" to use the value of the Unique\_Subject option.  
*project* is a project to use for the connection, or "" to use the value of the Project option.  
*server\_names* is a comma-delimited list of logical connection names to use for the connection, or "" to use the value of the Server\_Names option.  
*def\_subject\_prefix* is a default subject prefix to use for the connection, or "" to use the value of the Default\_Subject\_Prefix option.

**Return Type** TipcSrvConn

**C API** TipcSrvConnCreate

**Example** This example creates a connection that when opened has a unique subject of /accounting\_132, a project of "finance", will connect to the local machine, and prefix every subject with "/acctnt":

```
Dim SrvConn as TipcSrvConn
Dim Svc as New TipcSvc
Set SrvConn = Tsvc.TipcSrvConnCreate("/accounting_132", _
 "finance", _
 "tcp:_node", _
 "/acctnt")
```

## TipcSrvConnCreateNamed Method

The TipcSrvConnCreateNamed method creates a named RTclient connection handle to an RTserver. The name is associated with named options, which are set by the SetOption command or by using ParseStr. 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.

**Syntax** *object.TipcSrvConnCreateNamed name*  
where:  
*object* is an object expression that evaluates to a TipcSvc object.  
*name* is the name of the connection.

**Return Type** TipcSrvConn

**C API** TipcSrvConnCreateNamed

**Example** This example creates a named connection:

```
Dim SrvConn as TipcSrvConn
Dim Svc as New TipcSvc
Set SrvConn = Tsvc.TipcSrvConnCreateNamed("conn1")
```

## TMsgFileCreate Method

The TMsgFileCreate method creates a message file.

**Syntax** *object.TMsgFileCreate file\_name, io\_mode*  
where:  
*object* is an object expression that evaluates to a TipcSvc object.



*file\_name* is the name of the message file.

*io\_mode* is the mode used to create the message file. Valid values are:

- tipcMsgFileAppend
- tipcMsgFileRead
- tipcMsgFileWrite
- tipcMsgFileWriteBinary

|             |                                                                                                                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Return Type | None                                                                                                                                                                                          |
| C API       | TipcMsgFileCreate                                                                                                                                                                             |
| Example     | <p>This example creates a message file from an existing data.msg file:</p> <pre>Dim svc as New TipcSvc Dim mfile as TMsgFile set mfile = svc.TMsgFileCreate "data.msg", tipcMsgFileRead</pre> |

## XmlCreate Method

The XmlCreate method is similar to the Create Method in the TipcXml class. It creates an XML object containing a string of XML data.

**Syntax** *object.XmlCreate xml\_data*

where:

*object* is an object expression that evaluates to a TipcSvc object.

*xml\_data* is a string that specifies the XML data the returned TipcXml object contains.

|             |                                                                                                                                                                                              |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Return Type | TipcXml                                                                                                                                                                                      |
| C API       | TutXmlCreate                                                                                                                                                                                 |
| Example     | <p>This example creates a TipcXml object using TipcSvc:</p> <pre>Dim tXml As TipcXml Set tXml = Svc.XmlCreate("&lt;data&gt;&lt;data1&gt;&lt;temp=hot/&gt;&lt;/data1&gt;&lt;/data&gt;")</pre> |

## TipcSvc Class Properties

### DefaultMsgPriority Property

The DefaultMsgPriority property specifies the default priority of the messages sent through this connection. This priority can be overridden by setting the priority on individual messages or by using message types with different priorities.

**Syntax**     *object*.DefaultMsgPriority[=*priority*]  
where:  
*object*     is an object expression that evaluates to a TipcSvc object.  
*priority*    is an integer that specifies the default priority for messages sent on this connection.

**Data Type**    Long (Integer)

**C API**        Default\_Msg\_Priority option

**Example**      This example sets the default priority of all messages sent out with this connection to a value of 10:  
  
Svc.DefaultMsgPriority = 10

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## DefaultSubjectPrefix Property

---

The DefaultSubjectPrefix property specifies the default text prepended to subject names used by RTclient that do not begin with a slash (/).

**Syntax** `object.DefaultSubjectPrefix [=prefix]`

where:

*object* is an object expression that evaluates to a TipcSvc object.

*prefix* is a string that specifies the text to be prefixed to subjects that are not fully qualified, that is, that do not begin with a slash (/).

**Data Type** String

**Remarks** See the *TIBCO SmartSockets User's Guide* for detailed information about subject prefixes.

**C API** Default\_Subject\_Prefix option

**Example** This example adds "/StockTicker" to the beginning of each subject name that is not fully qualified:

```
Svc.DefaultSubjectPrefix = "/StockTicker"
```

## EnableControlMsgs Property

---

The EnableControlMsgs property specifies the commands allowed in a control message. See the *TIBCO SmartSockets User's Guide* for more information on control message enabling. The default value is "echo, quit."

**Syntax** `object.EnableControlMsgs [=commands]`

where:

*object* is an object expression that evaluates to a TipcSvc object.

*commands* is a string that specifies the commands allowed in a control message. The commands in the string must be separated by commas.

**Data Type** String

**C API** Enable\_Control\_Msgs option

**Example** This example shows the control messages enabled:

```
Dim s as String
s = Svc.EnableControlMsgs
If (s <> "echo, quit") Then
 MsgBox "Possible security hole!", 0, "Security
 Warning!"
End If
```

## LastErrorNum Property

---

The LastErrorNum property returns the global SmartSockets error number. This is a read-only value.

**Syntax** *object*.LastErrorNum  
where *object* is an object expression that evaluates to a TipcSvc object.

**Return Type** Long (Integer)

**C API** TutErrNumGet

**Example** This example returns the last SmartSockets error number:

```
Dim error_num As Long
error_num = Tipc.LastErrorNum
```

## LastErrorStr Property

---

The LastErrorStr property returns the global SmartSockets error number as a descriptive string. This is a read-only value.

**Syntax** *object*.LastErrorStr  
where *object* is an object expression that evaluates to a TipcSvc object.

**Return Type** String

**C API** TutErrStrGet

**Example** This example returns the last SmartSockets error string:

```
Dim error_str As String
error_str = Tipc.LastErrorStr
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## MonitorScope Property

The MonitorScope property specifies the level of interest for SmartSockets monitoring in those monitoring categories with no parameters (for example, RTclient names poll) or a parameter of @ (for example, RTclient time watch). MonitorScope acts as a filter, allowing a subset of the subject namespace to be specified. This is useful for RTclients monitoring only a subset of a large project.

|                  |                                                                                                                                                                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>    | <code>object.MonitorScope [=monitor_scope]</code><br>where:<br><code>object</code> is an object expression that evaluates to a TipcSvc object.<br><code>monitor_scope</code> is a string that specifies the monitoring scope of the RTclient. |
| <b>Data Type</b> | String                                                                                                                                                                                                                                        |
| <b>Remarks</b>   | For more information on monitoring scope, see the <i>TIBCO SmartSockets User's Guide</i> .                                                                                                                                                    |
| <b>C API</b>     | Monitor_Scope option                                                                                                                                                                                                                          |
| <b>Example</b>   | This example monitors only the cars subjects in the subject namespace:<br><code>Svc.MonitorScope = "/stocks/cars/..."</code>                                                                                                                  |

## ServerAutoFlushSize Property

The ServerAutoFlushSize property specifies the number of bytes buffered before data is automatically sent to an RTserver.

|                  |                                                                                                                                                                                                                                                                                   |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>    | <code>object.ServerAutoFlushSize [=flush_size]</code><br>where:<br><code>object</code> is an object expression that evaluates to a TipcSvc object.<br><code>flush_size</code> is a long integer that specifies the number of bytes to buffer before flushing data to an RTserver. |
| <b>Data Type</b> | Long (Integer)                                                                                                                                                                                                                                                                    |
| <b>C API</b>     | Server_Auto_Flush_Size option                                                                                                                                                                                                                                                     |

**Example** This example prints out the value of this property:

```
Debug.Print "AutoFlushSize value is: " &
 Format(Svc.ServerAutoFlushSize)
```

## ServerDeliveryTimeout Property

The `ServerDeliveryTimeout` property specifies the default maximum interval, in seconds, allowed for all receiving processes to acknowledge a message sent with GMD.

**Syntax** `object.ServerDeliveryTimeout[=timeout]`  
where:  
*object* is an object expression that evaluates to a `TipcSvc` object.  
*timeout* is a double real number that specifies the maximum GMD acknowledgement interval in seconds. The default value is 30.0.

**Data Type** Double

**C API** `Server_Delivery_Timeout` option

**Example** This example sets the server delivery timeout to 20 seconds:

```
Svc.ServerDeliveryTimeout = 20
If Svc.ServerDeliveryTimeout <> 20 Then
 Debug.Print "Unable to assign ServerDeliveryTimeout"
End If
```

## ServerKeepAliveTimeout Property

The `ServerKeepAliveTimeout` property specifies the maximum amount of time to wait, in seconds, when checking to see if the connection to an RTserver is still alive.

**Syntax** `object.ServerKeepAliveTimeout [=timeout]`  
where:  
*object* is an object expression that evaluates to a `TipcSvc` object.  
*timeout* is a double real number that specifies the maximum time in seconds to wait to determine if the RTserver is still alive. The default value is 5.0.

|                  |                                                                                                                                                                                                                                    |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Data Type</b> | Double                                                                                                                                                                                                                             |
| <b>C API</b>     | Server_Keep_Alive_Timeout option                                                                                                                                                                                                   |
| <b>Example</b>   | <p>This example sets the server keep alive timeout to 20 seconds:</p> <pre> Svc.ServerKeepAliveTimeout = 20 If Svc.ServerKeepAliveTimeout &lt;&gt; 20 Then     Debug.Print "Unable to assign ServerKeepAliveTimeout" End If </pre> |

## ServerMsgSend Property

The ServerMsgSend property specifies whether or not RTclient is allowed to send messages to an RTserver, or if messages are discarded without transmission. This is useful for backup processes that receive, but do not send messages.

|               |                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b> | <p><i>object</i>.ServerMsgSend [=enable_send]</p> <p>where:</p> <p><i>object</i> is an object expression that evaluates to a TipcSvc object.</p> <p><i>enable_send</i> is a Boolean value:</p> <ul style="list-style-type: none"> <li>• True allows the RTclient to transmit messages to the RTserver. This is the default value.</li> <li>• False does not allow the RTclient to send messages to the RTserver.</li> </ul> |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                  |                                                                                                                                                                                                                                                                                 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Data Type</b> | Boolean                                                                                                                                                                                                                                                                         |
| <b>C API</b>     | Server_Msg_Send option                                                                                                                                                                                                                                                          |
| <b>Example</b>   | <p>This example will not transmit any message with the MsgSendEx property:</p> <pre> Svc.ServerMsgSend = False SrvConn.MsgSend msg 'Msg will not be sent SrvConn.MsgSendEx msg, True 'Msg will not be sent SrvConn.MsgSendEx msg, False ' Msg will be sent SrvConn.Flush </pre> |

## ServerReadTimeout Property

The `ServerReadTimeout` property specifies how often, in seconds, data is expected to be available for reading on the connection to an RTserver. This setting is used to check for possible network failures.

**Syntax**     `object.ServerReadTimeout [=timeout]`  
where:  
*object*     is an object expression that evaluates to a `TipcSvc` object.  
*timeout*    is a double real number that specifies how often data is expected to be available for reading, in seconds. The default value is 30.0.

**Data Type**     Double

**C API**     `Server_Read_Timeout` option

**Example**     This example sets the server read timeout to 20 seconds:

```
Svc.ServerReadTimeout = 20
If Svc.ServerReadTimeout <> 20 Then
 Debug.Print "Unable to assign ServerReadTimeout"
End If
```

## ServerStartDelay Property

The `ServerStartDelay` property specifies the length of a delay, in seconds, between trying the different entries in the `ServerNames` property (when attempting to find or start an RTserver).

**Syntax**     `object.ServerStartDelay [=server_start_delay]`  
where:  
*object*             is an object expression that evaluates to a `TipcSvc` object.  
*server\_start\_delay* is a double real number that specifies the delay, in seconds, between trying the values in the `ServerNames` property. The default value is 1.0.

**Data Type**     Double

**C API**     `Server_Start_Delay` option



**Example** This example sets the delay to 10 seconds, if not already set:

```
If Svc.ServerStartDelay <> 10 Then
 Svc.ServerStartDelay = 10
End If
```

## ServerStartMaxTries Property

---

The ServerStartMaxTries property specifies how many times to traverse each connection, as specified by the entries in the ServerNames property.

**Syntax** *object*.ServerStartMaxTries [=start\_max\_tries]

where:

*object* is an object expression that evaluates to a TipSvc object.

*start\_max\_tries* is an integer that specifies the number of times to try connecting to an RTserver using the values in the ServerNames property. The default is 1.

**Data Type** Long (Integer)

**C API** Server\_Start\_Max\_Tries option

**Example** This example allows for three traversals of the ServerNames property list:

```
Svc.ServertStartMaxTries = 3
```

## ServerStartTimeout Property

---

The ServerStartTimeout property specifies the maximum time, in seconds, for the ActiveX components to wait for an RTserver to finish initializing.

**Syntax** *object*.ServerStartTimeout [=timeout]

where:

*object* is an object expression that evaluates to a TipSvc object.

*timeout* is a double real number that specifies the delay in seconds between trying the values in the ServerNames property. The default value is 30.0.

**Data Type** Double

**C API**     `Server_Start_Timeout` option

**Example**     This example sets the server start timeout to 20 seconds:

```
Svc.ServerStartTimeout = 20
If Svc.ServerStartTimeout <> 20 Then
 Debug.Print "Unable to assign ServerStartTimeout"
End If
```

## ServerWriteTimeout Property

---

The `ServerWriteTimeout` property specifies how long, in seconds, to block the ActiveX controls from writing data to an RTserver (before initiating failure recovery). If this timeout elapses between reads, a keep-alive is sent to the RTserver.

**Syntax**     `object.ServerWriteTimeout [=timeout]`  
where:  
*object*     is an object expression that evaluates to a `TipcSvc` object.  
*timeout*     is a double real number that specifies how long to block, in seconds, while writing to the RTserver. The default value is 30.0.

**Data Type**     Double

**C API**     `Server_Write_Timeout` option

**Example**     This example sets the server write timeout to 20 seconds:

```
Svc.ServerWriteTimeout = 20
If Svc.ServerWriteTimeout <> 20 Then
 Debug.Print "Unable to assign ServerWriteTimeout"
End If
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## TipcXml Class

---

TipcXml — user class for managing XML objects.

|                    |                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | The TipcXml class provides an interface to manage XML strings.                                                  |
| <b>Methods</b>     | <ul style="list-style-type: none"><li>• Clone Method</li><li>• Create Method</li><li>• Destroy Method</li></ul> |
| <b>Properties</b>  | <ul style="list-style-type: none"><li>• XmlString Property</li></ul>                                            |
| <b>Events</b>      | None                                                                                                            |

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## TipcXml Class Methods

---

### Clone Method

---

The Clone method makes a copy of an existing XML object. The copied XML object does not share any memory with the original XML object.

**Syntax**     *object*.Clone  
where *object* is an object expression that evaluates to a TipcXml object.

**Return Type**     TipcXml

**C API**     TutXmlClone

**Example**     This example clones an XML object:

```
Dim tXml As New TipcXml
Dim tXml_copy As New TipcXml
tXml.Create("<data><data1><temp=hot/></data1></data>")
Set tXml_copy = tXml.Clone
```

### Create Method

---

The Create method defines a new SmartSockets XML object. You should use the XmlCreate Method in the TipcSvc class to do this.

### Destroy Method

---

The Destroy method destroys a TipcXml object.

**Syntax**     *object*.Destroy  
where *object* is an object expression that evaluates to a TipcXml object.

**Return Type**     None

**C API**     TutXmlDestroy

**Example** This example destroys a TipcXml object:

```
Dim tXml As New TipcXml
Dim tXml_copy As New TipcXml
tXml.Create("<data><data1><temp=hot/></data1></data>")
Set tXml_copy = tXml.Clone
tXml_copy.Destroy
```

# TipcXml Class Properties

## XmlString Property

The XmlString property sets or returns value of the XML string that this object contains.

**Syntax**     *object*.XmlString [=*xml\_string*]  
              where:  
              *object*        is an object expression that evaluates to a TipcXml object.  
              *xml\_string*    is a string specifying the XML string that will replace the current string the object contains.

**Data Type**     String

**C API**         TutXmlSetStr, TutXmlGetStr

**Example**        This example replaces the string in an XML object:

```
Dim tXml As TipcXml
Dim svc as new TipcSvc
Dim tXml_copy As TipcXml
Set tXml = svc.XmlCreate("<data><data1><temp=hot/></data1></data>")
tXml.str = "<data><temp=cold/>"
Debug.Print tXml.str
```

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## TMsgFile Class

---

TMsgFile — user class that provides an interface for creating and reading message files.

|                    |                                                                                                                                       |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | The TMsgFile class provides an interface to create new messages files, or to read existing messages files.                            |
| <b>Methods</b>     | <ul style="list-style-type: none"><li>• Create Method</li><li>• Destroy Method</li><li>• Read Method</li><li>• Write Method</li></ul> |
| <b>Properties</b>  | None                                                                                                                                  |
| <b>Events</b>      | None                                                                                                                                  |

ALL INTERFACES IN THIS MANUAL ARE DEPRECATED

## TMsgFile Class Methods

---

### Create Method

---

The Create method creates a message file.

**Syntax**     *object*.Create *file\_name*, *io\_mode*  
              where:  
              *object*     is an object expression that evaluates to a TMsgFile object.  
              *file\_name* is the name of the message file.  
              *io\_mode*    is the mode used to create the message file. Valid values are:

- tipcMsgFileAppend
- tipcMsgFileRead
- tipcMsgFileWrite
- tipcMsgFileWriteBinary

**Return Type**     None

**C API**            TipcMsgFileCreate

**Example**        This example creates a new message file, data.msg, for writing:

```
Dim svc as New TipcSvc
Dim mfile as New TMsgFile
mfile.Create "data.msg", tipcMsgFileWrite
```

### Destroy Method

---

The Destroy method destroys a message file.

**Syntax**     *object*.Destroy  
              where *object* is an object expression that evaluates to a TMsgFile object.

**Return Type**     None

**C API**            TipcMsgFileDestroy



**Example** This example destroys the `data.msg` `msg_file` after it has been created:

```
Dim svc as New TipcSvc
Dim mfile as New TMsgFile
mfile.Create "data.msg", tipcMsgFileWrite
...
mfile.Destroy
```

## Read Method

The Read method reads a message from a message file.

**Syntax** `object.Read`

where *object* is an object expression that evaluates to a TMsgFile object.

**Return Type** None

**C API** TipcMsgFileRead

**Example** This example reads messages from the `data.msg` message file:

```
Dim svc as New TipcSvc
Dim mfile as New TMsgFile
Dim msg As TipcMsg

mfile.Create "file.msg", tipcMsgFileRead
Set msg = mfile.Read
```

## Write Method

The Write method writes a message to a message file.

**Syntax** `object.Write msg`

where:

*object* is an object expression that evaluates to a TMsgFile object.

*msg* is a TipcMsg object.

**Return Type** None

**C API** TipcMsgFileWrite

**Example** This example creates a message and writes to a message file:

```
Dim svc as New TipcSvc
Dim mt As TipcMt
Dim msg as TipcMsg
Dim mfile as New TMsgFile

mfile.Create "data.msg", tipcMsgFileWrite
set mt = svc.MtLookup("info")
set msg = svc.MsgCreate(mt)
msg.dest = "/system/thermal"
msg.appendStr "Value is in normal limits"
mfile.Write msg
```

## COM Error Handling

The SmartSockets COM interface uses COM Error Handling extensively through ISupportErrorInfo. In most cases, exceptions are raised when a method fails. Some methods provide other ways to detect failure, such as the MsgNext Method of the TipcSrvConn class, which returns an empty message in the event of a timeout. To handle these exceptions, calling applications should use this to identify the causes of errors.

The Error object populated contains:

|                 |                                                                                                                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Err.Description | The function name, a description of the error including the SmartSockets error if the error occurred inside a SmartSockets API call, and the location in the COM library to assist in customer support. |
| Err.Number      | The HRESULT of the offending COM call.                                                                                                                                                                  |
| Err.Source      | The class of the offending COM call.                                                                                                                                                                    |

### Error Handling Example

Errors can be handled by creating an error handler demonstrated below:

```
Sub ConnectAndSendMsg()

Dim svc As New TipcSvc
Dim msg As TipcMsg
Dim conn As TipcSrvConn

On Error GoTo Catch_Error:

Set msg = svc.MsgCreate(svc.MtLookup("info"))
' note we're creating a SrvConn to connect to a machine that does not
exist
Set conn = svc.TipcSrvConnCreate("/test1", "rtworks",
"tcp:doesnt_exist", "")
conn.Open

msg.AppendInt4 123
conn.MsgSend msg
conn.Flush

conn.Close
conn.Destroy

Catch_Error:
 Debug.Print "*****ERROR*****"
 Debug.Print "Description: " + Err.Description
 Debug.Print "Number: " + Str(Err.Number)
 Debug.Print "Source: " + Err.Source
 Debug.Print "***ERROR END***"
 Exit Sub

End Sub
```

Here is the result of running this function in the debug window. This error information is the same information returned through `TutErrNumGet` and `TutErrStrGet` that would be obtained after a C API call to `TipcSrvConnOpen`:

```
*****ERROR*****
Description: server not connected (502) in ITipcSrvConn::Open at
C:\shared\users\cls\ss62_11\cpsi\src\com\TipcSrvConn.cpp 163
Number: -2147221002
Source: TipcCom.TipcSrvConn.1
ERROR END
```

# Index

## A

AboutBox TipcSvc method 232  
 Accept TipcSrv method 191  
 Ack TipcMsg method 119, 292  
 ActiveX classes  
   overview 14, 72  
   TipcErrorCb 74  
   TipcMon 76  
   TipcMsg 117  
   TipcMsgFields 170  
   TipcMt 175  
   TipcProcessCb 186  
   TipcSrv 189  
   TipcSubjects 226  
   TipcSvc 231  
   TipcXml 257  
 ActiveX Components (loading) 3  
 ActiveX control icons 6  
 ActiveX example programs  
   see "example programs"  
 Add TipcMsgFields method 171, 378  
 Add TipcSubjects method 227, 534  
 AddDefaultCb TipcSrv method 191  
 AddErrorCb TipcSrv method 192  
 AddNamedPersistedObject TipcMsg method 298  
 AddNamedTimestamp TipcMsg method 304  
 AddNamedTimestampArray TipcMsg method 304  
 AddProcessCb TipcSrv method 193  
 AddSubjectCb TipcSrv method 194  
 AddSubjectDefaultCb TipcSrv method 195  
 AddUnknown TipcMsgFields method 172  
 AppendBinary TipcMsg method 128, 306  
 AppendChar TipcMsg method 129, 306  
 AppendInt2 TipcMsg method 129, 307  
 AppendInt2Array TipcMsg method 130, 307  
 AppendInt4 TipcMsg method 130, 308  
 AppendInt4Array TipcMsg method 131, 309  
 AppendInt8 TipcMsg method 309

AppendInt8Array TipcMsg method 310  
 AppendMsg TipcMsg method 132, 310  
 AppendMsgArray TipcMsg method 132, 311  
 AppendPersistedObject TipcMsg method 312  
 AppendReal16 TipcMsg method 314  
 AppendReal16Array TipcMsg method 315  
 AppendReal4 TipcMsg Method 133, 312  
 AppendReal4Array TipcMsg method 133, 313  
 AppendReal8 TipcMsg method 134, 313  
 AppendReal8Array TipcMsg method 135, 314  
 AppendStr TipcMsg method 135, 316  
 AppendStrArray TipcMsg method 136, 316, 318  
 AppendTimestamp TipcMsg method 317  
 AppendTimestampArray TipcMsg method 317  
 AppendUnknown TipcMsgFields method 379  
 AppendXML TipcMsg method 319  
 Arch TipcSrv property 217  
 Arch TipcSrvConn property 436  
 ArrivalTimestamp TipcMsg property 365  
 AutoFlushSize TipcSrv property 217  
 AutoFlushSize TipcSrvConn property 436  
 AutoMainLoop TipcSvc property 238

## B

BlockMode TipcSrv property 218  
 BlockMode TipcSrvConn property 437

## C

case sensitivity xxvii  
   on UNIX and Windows xxvii  
 CbDestroy TipcSrvConn method 401  
 Check TipcSrv method 196  
 Check TipcSrvConn method 401

- CheckMode TipcEvent property 280
- CheckVersion TipcSvc method 232
- ClientBufferGetWatch TipcMon method 77
- ClientBufferGetWatch TipcSrvMon method 466
- ClientBufferPoll TipcMon method 78
- ClientBufferPoll TipcSrvMon method 467
- ClientBufferSetWatch TipcMon method 79
- ClientBufferSetWatch TipcSrvMon method 468
- ClientCbPoll TipcMon method 79
- ClientCbPoll TipcSrvMon method 469
- ClientCongestionGetWatch TipcSrvMon method 470
- ClientCongestionSetWatch TipcSrvMon method 471
- ClientCpuPoll TipcSrvMon method 472
- ClientExtPoll TipcSrvMon method 473
- ClientGeneralPoll TipcMon method 80
- ClientGeneralPoll TipcSrvMon method 474
- ClientInfoPoll TipcSrvMon method 475
- ClientMsgRecvGetWatch TipcMon method 82
- ClientMsgRecvGetWatch TipcSrvMon method 477
- ClientMsgRecvSetWatch TipcMon method 83
- ClientMsgRecvSetWatch TipcSrvMon method 478
- ClientMsgSendGetWatch TipcMon method 83
- ClientMsgSendGetWatch TipcSrvMon method 479
- ClientMsgSendSetWatch TipcMon method 84
- ClientMsgSendSetWatch TipcSrvMon method 480
- ClientMsgTrafficPoll TipcMon method 85
- ClientMsgTrafficPoll TipcSrvMon method 481
- ClientMsgTypePoll TipcMon method 86
- ClientMsgTypePoll TipcSrvMon method 482
- ClientNamesGetWatch TipcMon method 87
- ClientNamesGetWatch TipcSrvMon method 483
- ClientNamesNumPoll TipcSrvMon method 484
- ClientNamesPoll TipcMon method 88
- ClientNamesPoll TipcSrvMon method 484
- ClientNamesSetWatch TipcMon method 89
- ClientNamesSetWatch TipcSrvMon method 485
- ClientOptionPoll TipcMon method 90
- ClientOptionPoll TipcSrvMon method 486
- ClientSubjectPoll TipcMon method 91
- ClientSubjectPoll TipcSrvMon method 487
- ClientSubscribeGetWatch TipcMon method 92
- ClientSubscribeGetWatch TipcSrvMon method 489
- ClientSubscribeNumPoll TipcSrvMon method 490
- ClientSubscribePoll TipcMon method 93
- ClientSubscribePoll TipcSrvMon method 491
- ClientSubscribeSetWatch TipcMon method 94
- ClientSubscribeSetWatch TipcSrvMon method 492
- ClientTimeGetWatch TipcMon method 94
- ClientTimePoll TipcMon method 95
- ClientTimeSetWatch TipcMon method 96
- Clone TipcMsg method 137, 319
- Clone TipcXml method 258, 560
- Close TipcSrvConn method 402
- CloseCbCreate TipcSrvConn method 403
- COM classes
  - overview 46, 262
  - TipcConnChangeCb 264
  - TipcDispatcher 267
  - TipcErrorCb 271
  - TipcEvent 273
  - TipcEventCb 283
  - TipcEventMsgCb 286
  - TipcMsg 289
  - TipcMsgFields 377
  - TipcMt 382
  - TipcProcessCb 396
  - TipcSrvConn 399
  - TipcSrvConnCache 454
  - TipcSrvMon 463
  - TipcSrvMonExt 519
  - TipcSubjects 533
  - TipcSvc 538
  - TipcXml 559
  - TMsgFile 563
- COM error handling 567
- COM example programs
  - see "example programs"
- COM Library (loading) 5
- ConnChange TipcConnChangeCb event 265
- ConnStatus TipcSrv property 218
- conventions used in this manual xxv
- CorrelationID TipcMsg property 365
- Count TipcMsgFields property 173, 380
- Count TipcSubjects property 229, 536
- Create TipcDispatcher method 268
- Create TipcEvent method 274
- Create TipcMsg method 137
- Create TipcMt method 176, 383
- Create TipcSrv method 196
- Create TipcSrvConn method 403

- Create TpcSrvMon method 493
- Create TpcSrvMonExt method 520
- Create TpcXml method 258, 560
- Create TMsgFile method 564
- CreateDetached TpcDispatcher method 268
- CreateMsg TpcEvent method 275
- CreateMsgType TpcEvent method 276
- CreateNamed TpcSrvConn method 404
- CreateOption method 539
- CreateServer TpcSrv method 197
- CreateServer TpcSrvConn method 405
- CreateSocket TpcEvent method 277
- CreateTimer TpcEvent method 278
- Current TpcMsg property 160, 366
- CurrentType TpcMsg method 138, 320
- CurrentUnknown TpcMsg method 320
- CurrentUnknown TpcMsg property 366
- customer support xxviii

## D

- DataArrival TpcSvc event 254
- DataTpcEvent property 280
- DefaultCbCreate TpcSrvConn method 404
- DefaultMsgPriority TpcSvc property 239, 550
- DefaultProtocols TpcSrvConn property 437
- DefaultSubjectPrefix TpcSrvConn property 438
- DefaultSubjectPrefix TpcSvc property 239, 551
- Delete TpcSrvMonExt method 520
- DeliveryMode TpcMsg property 161, 367
- DeliveryMode TpcMt property 179, 388
- DeliveryTimeout TpcMsg property 162, 368
- DeliveryTimeout TpcMt property 180, 389
- Delphi
  - example ActiveX program 21
  - see also "example programs"
- Dest TpcMsg property 163, 369
- Destroy TpcDispatcher method 268
- Destroy TpcEvent method 278
- Destroy TpcMsg method 138
- Destroy TpcMt method 177, 383
- Destroy TpcSrv method 197
- Destroy TpcSrvConn method 405

- Destroy TpcSrvMon method 493
- Destroy TpcSrvMonExt method 521
- Destroy TpcXml method 259, 560
- Destroy TMsgFile method 564
- Dispatch TpcDispatcher method 269
- Dispatcher TpcEvent property 281
- DispatcherEvent TpcEventCb event 284
- DispatcherMsgEvent TpcEventMsgCb event 287

## E

- EnableControlMsgs TpcSvc property 240, 551
- error handling for COM 567
- ErrorCbCreate TpcSrvConn method 406
- example programs
  - simple ActiveX in Delphi 21
  - simple ActiveX in PowerBuilder 23
  - simple ActiveX in Visual Basic 15
  - simple COM in Visual Basic 47
  - SSChat ActiveX 28
  - SSChat COM 54
  - WhoWhere ActiveX 33
  - WhoWhere COM 59
- extension data, monitoring 427, 473, 519

## F

- Fields TpcMsg property 163, 369
- file names
  - specifying xxvii
- Flush TpcSrv method 197
- Flush TpcSrvConn method 407
- functions
  - case-sensitivity xxvii

**G**

GetBufferReadSize TpcSrvConn method 407  
 GetBufferWriteSize TpcSrvConn method 408  
 GetCompression TpcMsg method 320  
 GetCompression TpcMt method 384  
 GetEnv TpcSvc method 539  
 GetIdentStr TpcMon method 97  
 GetNamedPersistedObject TpcMsg method 327  
 GetNamedTimestamp TpcMsg method 332  
 GetNamedTimestampArray TpcMsg method 333  
 GetOption TpcSvc method 540  
 GetSubscribeLb TpcSrvConn method 409  
 GetTrafficBytesRecv TpcSrvConn method 410  
 GetTrafficBytesSent TpcSrvConn method 411  
 GetTrafficMsgsRecv TpcSrvConn method 412  
 GetTrafficMsgsSent TpcSrvConn method 413  
 GmdDir TpcSrv property 219  
 GmdFileCreate TpcSrv method 198  
 GmdFileCreate TpcSrvConn method 413  
 GmdFileDelete TpcSrv method 199  
 GmdFileDelete TpcSrvConn method 414  
 GMDFinish TpcSrvConnCache method 455  
 GMDFlush TpcSrvConnCache method 455  
 GMDInit TpcSrvConnCache method 456  
 GmdMaxSize TpcSrv property 220  
 GmdMaxSize TpcSrvConn property 438  
 GmdMsgDelete TpcSrv method 199  
 GmdMsgDelete TpcSrvConn method 415  
 GmdMsgResend TpcSrv method 200  
 GmdMsgServerDelete TpcSrv method 200  
 GmdMsgServerDelete TpcSrvConn method 415  
 GmdMsgStatus TpcSrv method 201  
 GmdMsgStatus TpcSrvConn method 416  
 GmdNumPending TpcSrv property 220  
 GmdNumPending TpcSrvConn property 439  
 GmdResend TpcSrv method 201  
 Grammar TpcMt property 180, 390  
 GroupNames TpcSrvConnCache property 460

**H**

HeaderStrEncode TpcMsg property 164, 370  
 HeaderStrEncode TpcMt property 181, 390

**I**

identifiers  
     case sensitivity xxvii  
 Init TpcSrvConnCache method 457  
 InputEventsStrategy TpcSvc property 240  
 installation  
     ActiveX Components 3  
     ActiveX with Delphi 8  
     ActiveX with PowerBuilder 10  
     ActiveX with Visual Basic 6  
     COM Library 5  
 Interval TpcEvent property 281  
 IsNull TpcMsg method 149  
 Item TpcMsgFields property 173, 380  
 Item TpcSubjects property 229, 536

**K**

KeepAlive TpcSrv method 202  
 KeepAlive TpcSrvConn method 416



**L**

LastErrorNum TpcSvc property 241, 552  
 LastErrorStr TpcSvc property 241, 552  
 LbMode TpcMsg property 165, 371  
 LbMode TpcMt property 182, 391  
 LbMode TpcSrvConnCache property 460  
 Lcn TpcSrv property 221  
 Load TpcMsg method 334  
 LogAddMt TpcSrv method 202  
 LogRemoveMt TpcSrv method 203  
 Lookup TpcMt method 177, 385  
 LookupByNum TpcMt method 178, 386

**M**

MainLoop TpcDispatcher method 269  
 MainLoop TpcSrv method 204  
 MainLoop TpcSrvConn method 417  
 messages  
     case sensitivity xxvii  
 MonitorScope TpcSrvConn property 439  
 MonitorScope TpcSvc property 242, 553  
 MsgCreate TpcSvc method 233, 541  
 MsgInsert TpcSrv method 205  
 MsgInsert TpcSrvConn method 418  
 MsgNext TpcSrv method 206  
 MsgNext TpcSrvConn method 418  
 MsgProcess TpcSrv method 206  
 MsgProcess TpcSrvConn method 419  
 MsgSearchType TpcSrv method 207  
 MsgSearchType TpcSrvConn method 419  
 MsgSend TpcSrv method 208  
 MsgSend TpcSrvConn method 420  
 MsgSendEx TpcSrv method 208  
 MsgSendRpc TpcSrv method 209  
 MsgSendRpc TpcSrvConn method 421  
 MtCreate TpcSvc method 233, 542  
 MtLookup TpcSvc method 234, 543  
 MtLookupByNum TpcSvc method 543

**N**

Name TpcMt property 183, 393  
 NextBinary TpcMsg method 149, 336  
 NextChar TpcMsg method 150, 336  
 NextInt2 TpcMsg method 150, 337  
 NextInt2Array TpcMsg method 151, 337  
 NextInt4 TpcMsg method 152, 338  
 NextInt4Array TpcMsg method 152, 338  
 NextInt8 TpcMsg method 339  
 NextInt8Array TpcMsg method 339  
 NextMsg TpcMsg method 153, 340  
 NextMsgArray TpcMsg method 154, 341  
 NextPersistedObject TpcMsg method 341  
 NextReal16 TpcMsg method 344  
 NextReal16Array TpcMsg method 345  
 NextReal4 TpcMsg method 155, 342  
 NextReal4Array TpcMsg method 155, 342  
 NextReal8 TpcMsg method 156, 343  
 NextReal8Array TpcMsg method 156, 343  
 NextStr TpcMsg method 157, 345  
 NextStrArray TpcMsg method 158, 346  
 NextTimestamp TpcMsg method 346  
 NextTimestampArray TpcMsg method 347  
 NextType TpcMsg method 159, 348  
 NextUnknown TpcMsg method 159, 348  
 NextXml TpcMsg method 348  
 Node TpcSrv property 221  
 Node TpcSrvConn property 440  
 Num TpcMt property 183, 393  
 NumFields TpcMsg property 166, 372  
 NumQueued TpcSrv property 222  
 NumQueued TpcSrvConn property 441

**O**

Open TpcSrvConn method 422  
 OpenCbCreate TpcSrvConn method 422  
 Option TpcSvc property 243  
 options  
     case sensitivity xxvii

**P**

ParseFile TpcSvc method 235, 544  
 ParseStdLicense TpcSvc method 235  
 ParseStr TpcSvc method 236, 545  
 Pid TpcSrv property 222  
 Pid TpcSrvConn property 441  
 PowerBuilder  
     example ActiveX program 23  
     see also "example programs"  
 Priority TpcMsg property 166, 373  
 Priority TpcMt property 184, 394  
 ProcessCbCreate TpcSrvConn method 423  
 programs  
     see "example programs"  
 Project TpcSrvConn property 442  
 Project TpcSvc property 243  
 ProjectNamesGetWatch TpcMon method 97  
 ProjectNamesGetWatch TpcSrvMon method 494  
 ProjectNamesPoll TpcMon method 98  
 ProjectNamesPoll TpcSrvMon method 495  
 ProjectNamesSetWatch TpcMon method 99  
 ProjectNamesSetWatch TpcSrvMon method 496  
 PutEnv TpcSvc method 545

**R**

Read TMsgFile method 565  
 Remove TpcSubjects method 228, 535  
 RemoveAll TpcSubjects method 535  
 RemoveDefaultCb TpcSrv method 210  
 RemoveErrorCb TpcSrv method 211  
 RemoveProcessCb TpcSrv method 211  
 RemoveSubjectCb TpcSrv method 212  
 RemoveSubjectCbAll TpcSrv method 213  
 RemoveSubjectDefaultCb TpcSrv method 213  
 ReplyTo TpcMsg property 373  
 RequestMsgs TpcSrvConnCache method 457  
 RequestStart TpcSrvConnCache method 458  
 RequestStop TpcSrvConnCache method 459  
 RequestTimeout TpcSrvConnCache property 461  
 Reset TpcMsg method 349

**S**

sample programs  
     see "example programs"  
 Save TpcMsg method 349  
 Sender TpcMsg property 167, 374  
 SeqNum TpcMsg property 168, 374  
 ServerAutoConnect TpcSrvConn property 442  
 ServerAutoConnect TpcSvc property 244  
 ServerAutoFlushSize TpcSrvConn property 443  
 ServerAutoFlushSize TpcSvc property 244, 553  
 ServerBufferPoll TpcMon method 100  
 ServerBufferPoll TpcSrvMon method 496  
 ServerCongestionGetWatch TpcSrvMon method 498  
 ServerCongestionSetWatch TpcSrvMon method 499  
 ServerConnGetWatch TpcMon method 101  
 ServerConnGetWatch TpcSrvMon method 500  
 ServerConnPoll TpcMon method 101  
 ServerConnPoll TpcSrvMon method 500  
 ServerConnSetWatch TpcMon method 102  
 ServerConnSetWatch TpcSrvMon method 502  
 ServerCpuPoll TpcSrvMon method 502  
 ServerDeliveryTimeout TpcSrvConn property 443  
 ServerDeliveryTimeout TpcSvc property 245, 554  
 ServerGeneralPoll TpcMon method 103  
 ServerGeneralPoll TpcSrvMon method 503  
 ServerKeepAliveTimeout TpcSrvConn property 444  
 ServerKeepAliveTimeout TpcSvc property 245, 554  
 ServerMaxClientLicensesGetWatch TpcSrvMon  
     method 505  
 ServerMaxClientLicensesSetWatch TpcSrvMon  
     method 506  
 ServerMaxReconnectDelay TpcSrvConn  
     property 444  
 ServerMsgSend TpcSrvConn property 445  
 ServerMsgSend TpcSvc property 246, 555  
 ServerMsgTrafficPoll TpcMon method 104  
 ServerMsgTrafficPoll TpcSrvMon method 507  
 ServerNames TpcSrvConn property 446  
 ServerNames TpcSvc property 247  
 ServerNamesGetWatch TpcMon method 105  
 ServerNamesGetWatch TpcSrvMon method 508  
 ServerNamesPoll TpcMon method 106  
 ServerNamesPoll TpcSrvMon method 508  
 ServerNamesSetWatch TpcMon method 107

- ServerNamesSetWatch TpcSrvMon method 509
- ServerOptionPoll TpcMon method 108
- ServerOptionPoll TpcSrvMon method 510
- ServerReadTimeout TpcSrvConn property 446
- ServerReadTimeout TpcSvc property 247, 556
- ServerRoutePoll TpcMon method 109
- ServerRoutePoll TpcSrvMon method 511
- ServerStartDelay TpcSrvConn property 447
- ServerStartDelay TpcSvc property 248, 556
- ServerStartMaxTimeout TpcSrvConn property 447
- ServerStartMaxTries TpcSrvConn property 448
- ServerStartMaxTries TpcSvc property 248, 557
- ServerStartTimeout TpcSvc property 249, 557
- ServerTimePoll TpcMon method 110
- ServerTimePoll TpcSrvMon method 512
- ServerWriteTimeout TpcSrvConn property 448
- ServerWriteTimeout TpcSvc property 249, 558
- SetBinary TpcSrvMon method 521
- SetBoolArray TpcSrvMon method 522
- SetBoolTpcSrvMon method 522
- SetCompression TpcMsg method 350
- SetCompression TpcMt method 386
- SetCredentials TpcSrvConn method 424
- SetdentStr TpcMon method 111
- SetInt2 TpcSrvMon method 523
- SetInt2Array TpcSrvMon method 524
- SetInt4 TpcSrvMon method 524
- SetInt4Array TpcSrvMon method 525
- SetInt8 TpcSrvMon method 525
- SetInt8Array TpcSrvMon method 526
- SetOption TpcSvc method 546
- SetReal16 TpcSrvMon method 529
- SetReal16Array TpcSrvMon method 529
- SetReal4 TpcSrvMon method 527
- SetReal4Array TpcSrvMon method 527
- SetReal8 TpcSrvMon method 528
- SetReal8Array TpcSrvMon method 528
- SetStdSubjectSubscribe TpcSrvConn method 425
- SetStr TpcSrvMon method 530
- SetStrArray TpcSrvMon method 531
- SetUsernamePassword TpcSrvConn method 425
- SetUtf8 TpcSrvMon method 531
- SetUtf8Array TpcSrvMon method 532
- shell commands
  - specifying xxvii
- Socket TpcEvent property 282
- SocketConnectTimeout TpcSvc property 250
- Srv TpcSvc property 251
- SrvCreate TpcSvc event 254
- SrvDestroy TpcSvc event 256
- SrvIsRunning TpcSrvConn method 426
- SrvLock TpcSrvConn method 426
- SrvUnlock TpcSrvConn method 428
- SSChat example program
  - ActiveX 28
  - COM 54
- Status TpcSrvConn property 449
- StdSubjectRetrieve TpcSrv method 214
- StdSubjectSubscribe TpcSrv method 214
- SubjectCbCreate TpcSrvConn method 428
- SubjectCbDestroyAll TpcSrvConn method 429
- SubjectDefaultCbCreate TpcSrvConn method 430
- SubjectGmdInit TpcSrv method 215
- SubjectGmdInit TpcSrvConn method 431
- SubjectLbInit TpcSrv method 216
- SubjectLbInit TpcSrvConn method 431
- SubjectNamesGetWatch TpcMon method 111
- SubjectNamesGetWatch TpcSrvMon method 513
- SubjectNamesPoll TpcMon method 112
- SubjectNamesPoll TpcSrvMon method 514
- SubjectNamesSetWatch TpcMon method 113
- SubjectNamesSetWatch TpcSrvMon method 515
- SubjectRetrieve TpcSrv method 216
- Subjects TpcSrv property 223
- Subjects TpcSrvConn property 450
- Subjects TpcSvc property 251
- SubjectSetSubscribe TpcSrvConn method 432
- SubjectSetSubscribeEx TpcSrvConn method 433
- SubjectSetSubscribeLb TpcSrvConn method 433
- SubjectSubscribeGetWatch TpcMon method 113
- SubjectSubscribePoll TpcMon method 114
- SubjectSubscribeSetWatch TpcMon method 115
- SubscribeGetWatch TpcSrvMon method 515
- SubscribePoll TpcSrvMon method 516
- SubscribeSetWatchTpcSrvMon method 517
- support, contacting xxviii

T

technical support xxviii

Timeout TpcSrv property 223

Timeout TpcSrvConn property 451

TpcConnChangeCb COM class 264

TpcConnChangeCb events

    ConnChange 265

TpcDispatcher COM class 267

TpcDispatcher methods

    Create 268

    CreateDetached 268

    Destroy 268

    Dispatch 269

    MainLoop 269

TpcDispatcherCreate TpcSvc method 546

TpcDispatcherCreateDetached TpcSvc method 547

TpcError TpcErrorCb event 74, 271

TpcErrorCb ActiveX class 74

TpcErrorCb COM class 271

TpcErrorCb events

    TpcError 74, 271

TpcEvent COM class 273

TpcEvent methods

    Create 274

    CreateMsg 275

    CreateMsgType 276

    CreateSocket 277

    CreateTimer 278

    Destroy 278

TpcEvent properties

    CheckMode 280

    Data 280

    Dispatcher 281

    Interval 281

    Socket 282

    Type 282

TpcEventCb COM class 283

TpcEventCb events

    DispatcherEvent 284

TpcEventMsgCb COM class 286

TpcEventMsgCb events

    DispatcherMsgEvent 287

TpcMon ActiveX class 76

TpcMon methods

ClientBufferGetWatch 77

ClientBufferPoll 78

ClientBufferSetWatch 79

ClientCbPoll 79

ClientGeneralPoll 80

ClientMsgRecvGetWatch 82

ClientMsgRecvSetWatch 83

ClientMsgSendGetWatch 83

ClientMsgSendSetWatch 84

ClientMsgTrafficPoll 85

ClientMsgTypePoll 86

ClientNamesGetWatch 87

ClientNamesPoll 88

ClientNamesSetWatch 89

ClientOptionPoll 90

ClientSubjectPoll 91

ClientSubscribeGetWatch 92

ClientSubscribePoll 93

ClientSubscribeSetWatch 94

ClientTimeGetWatch 94

ClientTimePoll 95

ClientTimeSetWatch 96

GetIdentStr 97

ProjectNamesGetWatch 97

ProjectNamesPoll 98

ProjectNamesSetWatch 99

ServerBufferPoll 100

ServerConnGetWatch 101

ServerConnPoll 101

ServerConnSetWatch 102

ServerGeneralPoll 103

ServerMsgTrafficPoll 104

ServerNamesGetWatch 105

ServerNamesPoll 106

ServerNamesSetWatch 107

ServerOptionPoll 108

ServerRoutePoll 109

ServerTimePoll 110

SetdentStr 111

SubjectNamesGetWatch 111

SubjectNamesPoll 112

SubjectNamesSetWatch 113

SubjectSubscribeGetWatch 113

SubjectSubscribePoll 114

SubjectSubscribeSetWatch 115

- TipcMsg ActiveX class 117
- TipcMsg COM class 289
- TipcMsg methods
  - Ack 119, 292
  - AddNamedPersistedObject 298
  - AddNamedTimestamp 304
  - AddNamedTimestampArray 304
  - AppendBinary 128, 306
  - AppendChar 129, 306
  - AppendInt2 129, 307
  - AppendInt2Array 130, 307
  - AppendInt4 130, 308
  - AppendInt4Array 131, 309
  - AppendInt8 309
  - AppendInt8Array 310
  - AppendMsg 132, 310
  - AppendMsgArray 132, 311
  - AppendPersistedObject 312
  - AppendReal16 314
  - AppendReal16Array 315
  - AppendReal4 133, 312
  - AppendReal4Array 133, 313
  - AppendReal8 134, 313
  - AppendReal8Array 135, 314
  - AppendStr 135, 316
  - AppendStrArray 136, 316, 318
  - AppendTimestamp 317
  - AppendTimestampArray 317
  - AppendXML 319
  - Clone 137, 319
  - Create 137
  - CurrentType 138, 320
  - CurrentUnknown 320
  - Destroy 138
  - GetCompression 320
  - GetNamedPersistedObject 327
  - GetNamedTimestamp 332
  - GetNamedTimestampArray 333
  - IsNull 149
  - Load 334
  - NextBinary 149, 336
  - NextChar 150, 336
  - NextInt2 150, 337
  - NextInt2Array 151, 337
  - NextInt4 152, 338
  - NextInt4Array 152, 338
  - NextInt8 339
  - NextInt8Array 339
  - NextMsg 153, 340
  - NextMsgArray 154, 341
  - NextPersistedObject 341
  - NextReal16 344
  - NextReal16Array 345
  - NextReal4 155, 342
  - NextReal4Array 155, 342
  - NextReal8 156, 343
  - NextReal8Array 156, 343
  - NextStr 157, 345
  - NextStrArray 158, 346
  - NextTimestamp 346
  - NextTimestampArray 347
  - NextType 159, 348
  - NextUnknown 159, 348
  - NextXml 348
  - Reset 349
  - Save 349
  - SetCompression 350
  - UpdateNamedBinary 351
  - UpdateNamedCharacter 352
  - UpdateNamedInt2 352
  - UpdateNamedInt2Array 353
  - UpdateNamedInt4 353
  - UpdateNamedInt4Array 354
  - UpdateNamedInt8 354
  - UpdateNamedInt8Array 355
  - UpdateNamedMsg 356
  - UpdateNamedMsgArray 356
  - UpdateNamedPersistedObject 357
  - UpdateNamedReal16 360
  - UpdateNamedReal16Array 361
  - UpdateNamedReal4 358
  - UpdateNamedReal4Array 358
  - UpdateNamedReal8 359
  - UpdateNamedReal8Array 359
  - UpdateNamedStr 361
  - UpdateNamedStrArray 362
  - UpdateNamedTimestamp 362
  - UpdateNamedTimestampArray 363
  - UpdateNamedXML 364

- TipcMsg properties
  - ArrivalTimestamp 365
  - CorrelationID 365
  - Current 160, 366
  - CurrentUnknown 366
  - DeliveryMode 161, 367
  - DeliveryTimeout 162, 368
  - Dest 163, 369
  - Fields 163, 369
  - HeaderStrEncode 164, 370
  - LbMode 165, 371
  - NumFields 166, 372
  - Priority 166, 373
  - ReplyTo 373
  - Sender 167, 374
  - SeqNum 168, 374
  - Type 168, 375
  - UserProp 169, 376
- TipcMsg TipcProcessCb event 187, 397
- TipcMsgFields ActiveX class 170
- TipcMsgFields COM class 377
- TipcMsgFields methods
  - Add 171, 378
  - AddUnknown 172
  - AppendUnknown 379
- TipcMsgFields properties
  - Count 173, 380
  - Item 173, 380
- TipcMt ActiveX class 175
- TipcMt COM class 382
- TipcMt methods
  - Create 176, 383
  - Destroy 177, 383
  - GetCompression 384
  - Lookup 177, 385
  - LookupByNum 178, 386
  - SetCompression 386
- TipcMt properties
  - DeliveryMode 179, 388
  - DeliveryTimeout 180, 389
  - Grammar 180, 390
  - HeaderStrEncode 181, 390
  - LbMode 182, 391
  - Name 183, 393
  - Num 183, 393
  - Priority 184, 394
  - UserProp 184, 394
- TipcProcessCb ActiveX class 186
- TipcProcessCb COM class 396
- TipcProcessCb events
  - TipcMsg 187, 397
- TipcSrv ActiveX class 189
- TipcSrv methods
  - Accept 191
  - AddDefaultCb 191
  - AddErrorCb 192
  - AddProcessCb 193
  - AddSubjectCb 194
  - AddSubjectDefaultCb 195
  - Check 196
  - Create 196
  - CreateServer 197
  - Destroy 197
  - Flush 197
  - GmdFileCreate 198
  - GmdFileDelete 199
  - GmdMsgDelete 199
  - GmdMsgResend 200
  - GmdMsgServerDelete 200
  - GmdMsgStatus 201
  - GmdResend 201
  - KeepAlive 202
  - LogAddMt 202
  - LogRemoveMt 203
  - MainLoop 204
  - MsgInsert 205
  - MsgNext 206
  - MsgProcess 206
  - MsgSearchType 207
  - MsgSend 208
  - MsgSendEx 208
  - MsgSendRpc 209
  - RemoveDefaultCb 210
  - RemoveErrorCb 211
  - RemoveProcessCb 211
  - RemoveSubjectCb 212
  - RemoveSubjectCbAll 213
  - RemoveSubjectDefaultCb 213
  - StdSubjectRetrieve 214
  - StdSubjectSubscribe 214

- SubjectGmdInit 215
- SubjectLbInit 216
- SubjectRetrieve 216
- TipcSrv properties
  - Arch 217
  - AutoFlushSize 217
  - BlockMode 218
  - ConnStatus 218
  - GmdDir 219
  - GmdMaxSize 220
  - GmdNumPending 220
  - Lcn 221
  - Node 221
  - NumQueued 222
  - Pid 222
  - Subjects 223
  - Timeout 223
  - UniqueSubject 224
  - User 225
- TipcSrvConn COM class 399
- TipcSrvConn methods
  - CbDestroy 401
  - Check 401
  - Close 402
  - CloseCbCreate 403
  - Create 403
  - CreateNamed 404
  - CreateServer 405
  - DefaultCbCreate 404
  - Destroy 405
  - ErrorCbCreate 406
  - Flush 407
  - GetBufferReadSize 407
  - GetBufferWriteSize 408
  - GetSubscribeLb 409
  - GetTrafficBytesRecv 410
  - GetTrafficBytesSent 411
  - GetTrafficMsgsRecv 412
  - GetTrafficMsgsSent 413
  - GmdFileCreate 413
  - GmdFileDelete 414
  - GmdMsgDelete 415
  - GmdMsgServerDelete 415
  - GmdMsgStatus 416
  - KeepAlive 416
  - MainLoop 417
  - MsgInsert 418
  - MsgNext 418
  - MsgProcess 419
  - MsgSearchType 419
  - MsgSend 420
  - MsgSendRpc 421
  - Open 422
  - OpenCbCreate 422
  - ProcessCbCreate 423
  - SetCredentials 424
  - SetStdSubjectSubscribe 425
  - SetUsernamePassword 425
  - SrvIsRunning 426
  - SrvLock 426
  - SrvUnlock 428
  - SubjectCbCreate 428
  - SubjectCbDestroyAll 429
  - SubjectDefaultCbCreate 430
  - SubjectGmdInit 431
  - SubjectLbInit 431
  - SubjectSetSubscribe 432
  - SubjectSetSubscribeEx 433
  - SubjectSetSubscribeLb 433
  - TipcSrvMonCreate 427
  - TipcSrvMonExtCreate 427
- TipcSrvConn properties
  - Arch 436
  - AutoFlushSize 436
  - BlockMode 437
  - DefaultProtocols 437
  - DefaultSubjectPrefix 438
  - GmdMaxSize 438
  - GmdNumPending 439
  - MonitorScope 439
  - Node 440
  - NumQueued 441
  - Pid 441
  - Project 442
  - ServerAutoConnect 442
  - ServerAutoFlushSize 443
  - ServerDeliveryTimeout 443
  - ServerKeepAliveTimeout 444
  - ServerMaxReconnectDelay 444
  - ServerMsgSend 445

- ServerNames 446
- ServerReadTimeout 446
- ServerStartDelay 447
- ServerStartMaxTimeout 447
- ServerStartMaxTries 448
- ServerWriteTimeout 448
- Status 449
- Subjects 450
- Timeout 451
- UniqueSubject 452
- User 452
- XtSource 453
- TipcSrvConnCache COM class 454
- TipcSrvConnCache methods
  - GMDFinish 455
  - GMDFlush 455
  - GMDInit 456
  - Init 457
  - RequestMsgs 457
  - RequestStart 458
  - RequestStop 459
- TipcSrvConnCache properties
  - GroupNames 460
  - LbMode 460
  - RequestTimeout 461
- TipcSrvConnCreate TipcSvc method 547
- TipcSrvConnCreateNamed TipcSvc method 548
- TipcSrvMon COM class 463
- TipcSrvMon methods
  - ClientBufferGetWatch 466
  - ClientBufferPoll 467
  - ClientBufferSetWatch 468
  - ClientCbPoll 469
  - ClientCongestionGetWatch 470
  - ClientCongestionSetWatch 471
  - ClientCpulPoll 472
  - ClientExtPoll 473
  - ClientGeneralPoll 474
  - ClientInfoPoll 475
  - ClientMsgRecvGetWatch 477
  - ClientMsgRecvSetWatch 478
  - ClientMsgSendGetWatch 479
  - ClientMsgSendSetWatch 480
  - ClientMsgTrafficPoll 481
  - ClientMsgTypePoll 482
  - ClientNamesGetWatch 483
  - ClientNamesNumPoll 484
  - ClientNamesPoll 484
  - ClientNamesSetWatch 485
  - ClientOptionPoll 486
  - ClientSubjectPoll 487
  - ClientSubscribeGetWatch 489
  - ClientSubscribeNumPoll 490
  - ClientSubscribePoll 491
  - ClientSubscribeSetWatch 492
  - Create 493
  - Destroy 493
  - ProjectNamesGetWatch 494
  - ProjectNamesPoll 495
  - ProjectNamesSetWatch 496
  - ServerBufferPoll 496
  - ServerCongestionGetWatch 498
  - ServerCongestionSetWatch 499
  - ServerConnGetWatch 500
  - ServerConnPoll 500
  - ServerConnSetWatch 502
  - ServerCpulPoll 502
  - ServerGeneralPoll 503
  - ServerMaxClientLicensesGetWatch 505
  - ServerMaxClientLicensesSetWatch 506
  - ServerMsgTrafficPoll 507
  - ServerNamesGetWatch 508
  - ServerNamesPoll 508
  - ServerNamesSetWatch 509
  - ServerOptionPoll 510
  - ServerRoutePoll 511
  - ServerTimePoll 512
  - SubjectNamesGetWatch 513
  - SubjectNamesPoll 514
  - SubjectNamesSetWatch 515
  - SubscribeGetWatch 515
  - SubscribePoll 516
  - SubscribeSetWatch 517
- TipcSrvMonCreate TipcSrvConn method 427
- TipcSrvMonExt COM class 519
- TipcSrvMonExt methods
  - Create 520
  - Delete 520
  - Destroy 521
  - SetBinary 521



- SetBool 522
- SetBoolArray 522
- SetInt2 523
- SetInt2Array 524
- SetInt4 524
- SetInt4Array 525
- SetInt8 525
- SetInt8Array 526
- SetReal16 529
- SetReal16Array 529
- SetReal4 527
- SetReal4Array 527
- SetReal8 528
- SetReal8Array 528
- SetStr 530
- SetStrArray 531
- SetUtf8 531
- SetUtf8Array 532
- TipcSrvMonExtCreate TipcSrvConn method 427
- TipcSubjects ActiveX class 226
- TipcSubjects COM class 533
- TipcSubjects methods
  - Add 227, 534
  - Remove 228, 535
  - RemoveAll 535
- TipcSubjects properties
  - Count 229, 536
  - Item 229, 536
- TipcSvc ActiveX class 231
- TipcSvc COM class 538
- TipcSvc events
  - DataArrival 254
  - SrvCreate 254
  - SrvDestroy 256
- TipcSvc methods
  - AboutBox 232
  - CheckVersion 232
  - CreateOption 539
  - GetEnv 539
  - GetOption 540
  - MsgCreate 233, 541
  - MtCreate 233, 542
  - MtLookup 234, 543
  - MtLookupByNum 543
  - ParseFile 235, 544
  - ParseStdLicense 235
  - ParseStr 236, 545
  - PutEnv 545
  - SetOption 546
  - TipcDispatcherCreate 546
  - TipcDispatcherCreateDetached 547
  - TipcSrvConnCreate 547
  - TipcSrvConnCreateNamed 548
  - TMsgFileCreate 548
  - XmlCreate 236, 549
- TipcSvc properties
  - AutoMainLoop 238
  - DefaultMsgPriority 239, 550
  - DefaultSubjectPrefix 239, 551
  - EnableControlMsgs 240, 551
  - InputEventsStrategy 240
  - LastErrorNum 241, 552
  - LastErrorStr 241, 552
  - MonitorScope 242, 553
  - Option 243
  - Project 243
  - ServerAutoConnect 244
  - ServerAutoFlushSize 244, 553
  - ServerDeliveryTimeout 245, 554
  - ServerKeepAliveTimeout 245, 554
  - ServerMsgSend 246, 555
  - ServerNames 247
  - ServerReadTimeout 247, 556
  - ServerStartDelay 248, 556
  - ServerStartMaxTries 248, 557
  - ServerStartTimeout 249, 557
  - ServerWriteTimeout 249, 558
  - SocketConnectTimeout 250
  - Srv 251
  - Subjects 251
  - UdpBroadcastTimeout 252
  - UniqueSubject 252
- TipcXml ActiveX class 257
- TipcXml COM class 559
- TipcXml methods
  - Clone 258, 560
  - Create 258, 560
  - Destroy 259, 560
- TipcXml properties
  - XmlString 562

TMsgFile COM class 563  
TMsgFile methods  
    Create 564  
    Destroy 564  
    Read 565  
    Write 565  
TMsgFileCreate method 548  
Type TipcEvent property 282  
Type TipcMsg property 168, 375

U

UdpBroadcastTimeout TipcSvc property 252  
UniqueSubject TipcSrv property 224  
UniqueSubject TipcSrvConn property 452  
UniqueSubject TipcSvc property 252  
UpdateNamedBinary TipcMsg method 351  
UpdateNamedCharacter TipcMsg method 352  
UpdateNamedInt2 TipcMsg method 352  
UpdateNamedInt2Array TipcMsg method 353  
UpdateNamedInt4 TipcMsg method 353  
UpdateNamedInt4Array TipcMsg method 354  
UpdateNamedInt8 TipcMsg method 354  
UpdateNamedInt8Array TipcMsg method 355  
UpdateNamedMsg TipcMsg method 356  
UpdateNamedMsgArray TipcMsg method 356  
UpdateNamedPersistedObject TipcMsg method 357  
UpdateNamedReal16 TipcMsg method 360  
UpdateNamedReal16Array TipcMsg method 361  
UpdateNamedReal4 TipcMsg method 358  
UpdateNamedReal4Array TipcMsg method 358  
UpdateNamedReal8 TipcMsg method 359  
UpdateNamedReal8Array TipcMsg method 359  
UpdateNamedStr TipcMsg method 361  
UpdateNamedStrArray TipcMsg method 362  
UpdateNamedTimestamp TipcMsg method 362  
UpdateNamedTimestampArray TipcMsg method 363  
UpdateNamedXML TipcMsg method 364  
User TipcSrv property 225  
User TipcSrvConn property 452  
UserProp TipcMsg property 169, 376  
UserProp TipcMt property 184, 394

V

Visual Basic  
    example ActiveX program 15  
    example COM program 47  
    see also "example programs"

W

WhoWhere example program  
    ActiveX 33  
    COM 59  
Write TMsgFile method 565

X

XmlCreate TipcSvc method 236, 549  
XmlString TipcXml property 562  
XtSource TipcSrvConn property 453