

[MS-DTCO]: MSDTC Connection Manager: OleTx Transaction Protocol Specification

Intellectual Property Rights Notice for Protocol Documentation

- This protocol documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the protocols, and may distribute portions of it in your implementations of the protocols or your documentation as necessary to properly document the implementation. This permission also applies to any documents that are referenced in the protocol documentation.
- Microsoft does not claim any trade secret rights in this documentation.
- Microsoft has patents that may cover your implementations of the protocols. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. If you are interested in obtaining a patent license, please contact protocol@microsoft.com.
- The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them.

Revision Summary

Date	Revision History	Revision Class	Comments
04/03/2007	0.01		MCPD Milestone Longhorn Initial Availability
07/03/2007	1.0	Major	MLonghorn+90
07/20/2007	1.1	Minor	Minor work due to other OleTx protocols shipping.
08/10/2007	1.1.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
09/28/2007	1.2	Minor	Updated the technical content.
10/23/2007	2.0	Major	Updated and revised the technical content.
11/30/2007	2.1	Minor	Updated the technical content.
01/25/2008	2.1.1	Editorial	Revised and edited the technical content.

Table of Contents

1	Introduction	25
1.1	Glossary	25
1.2	References	28
1.2.1	Normative References	28
1.2.2	Informative References.....	29
1.3	Protocol Overview (Synopsis).....	29
1.3.1	Transaction Lifetime.....	30
1.3.1.1	Phase Zero	31
1.3.1.2	Phase One.....	32
1.3.1.3	Phase Two.....	34
1.3.2	Additional Considerations.....	36
1.3.2.1	Unilateral Abort.....	36
1.3.2.2	Single-Phase Commit.....	37
1.3.3	Transaction Roles	38
1.3.3.1	Application Role	38
1.3.3.2	Resource Manager Role	39
1.3.3.3	Transaction Manager Role.....	39
1.3.3.3.1	Core Transaction Manager Facet	41
1.3.3.3.2	Transaction Manager Communication with an Application Facet	41
1.3.3.3.3	Transaction Manager Communication with a Resource Manager Facet	42
1.3.3.3.4	Superior Transaction Manager Facet	42
1.3.3.3.5	Subordinate Transaction Manager Facet	42
1.3.4	Transaction Recovery	42
1.3.4.1	Relationship Between Recovery and Durability	43
1.3.4.2	Resource Manager Recovery	43
1.3.4.3	Transaction Manager Recovery	44
1.3.5	Transaction Propagation	44
1.3.5.1	Pull Propagation	45
1.3.5.2	Push Propagation	46
1.4	Relationship to Other Protocols.....	48
1.5	Prerequisites/Preconditions	49
1.6	Applicability Statement	49
1.7	Versioning and Capability Negotiation.....	49
1.7.1	Versioning Mechanisms.....	49
1.7.2	Versioning Negotiation Mechanisms	50
1.7.3	Capability Negotiation Mechanisms	50
1.8	Vendor-Extensible Fields	51
1.9	Standards Assignments.....	51
2	Messages	52
2.1	Transport.....	52
2.1.1	Messages, Connections, and Sessions	52
2.1.2	MS-CMPO Parameterization	52
2.1.2.1	Computing a Security Level	52
2.1.2.2	Computing Protocol Version Values	52
2.1.2.3	Computing a Name Object	53
2.2	Message Syntax	53
2.2.1	Protocol Versioning	53
2.2.1.1	Protocol Version Numbers as a Versioning Mechanism.....	53
2.2.1.1.1	Version-Specific Aspects of Connection Types Relevant to an Application.....	54
2.2.1.1.2	Version-Specific Aspects of Connection Types Relevant to a Transaction Manager	55

2.2.1.1.3	Version-Specific Aspects of Connection Types Relevant to a Resource Manager	55
2.2.2	Structures with Fields Containing Version Numbers as Versioning Mechanism	56
2.2.3	Structures with a Format-Specifying Field as Versioning Mechanism	56
2.2.4	Common Structures	56
2.2.4.1	MESSAGE_PACKET	56
2.2.4.2	OLETX_TM_ADDR.....	57
2.2.4.3	OLETX_VARLEN_STRING	58
2.2.5	Transaction Propagation Structures	59
2.2.5.1	Associate_Msg_Version2	59
2.2.5.2	Associate_Msg_Version3	59
2.2.5.3	NAMEOBJECTBLOB	60
2.2.5.4	Propagation_Token.....	62
2.2.5.5	SDtcCmEndpointInfoV1	64
2.2.5.6	SDtcCmEndpointInfoV2	64
2.2.5.7	SOleTxInfoForTip	65
2.2.5.8	SExtendedEndpointInfo	66
2.2.5.9	STmToTmProtocol	67
2.2.5.10	STxInfo	68
2.2.5.11	SWhereabouts	70
2.2.6	Transaction Enumerations.....	70
2.2.6.1	Connection Types	70
2.2.6.2	TM_Protocol	73
2.2.6.3	TXUSER_ENLISTMENT_PREPAREREQDONE_RESPONSE	73
2.2.6.4	PARTNERTM_PROPAGATE_PREPAREREQDONE_RESPONSE.....	73
2.2.6.5	TXUSER_VOTER_VOTERREQDONE_RESPONSE.....	74
2.2.6.6	TRUN_TXBEGIN_ERRORS	74
2.2.6.7	TRUN_TXIMPORT_ERRORS	75
2.2.6.8	OLETX_ISOLATION_FLAGS	75
2.2.6.9	OLETX_ISOLATION_LEVEL.....	77
2.2.7	Transaction Constants	77
2.2.7.1	GRFRM	77
2.2.7.2	DTCADVCONFIG.....	78
2.2.8	Connection Types Relevant to Applications	80
2.2.8.1	Transaction Initiation and Completion.....	80
2.2.8.1.1	CONNTYPE_TXUSER_BEGINNER.....	80
2.2.8.1.1.1	TXUSER_BEGINNER_MTAG_ABORT	80
2.2.8.1.1.2	TXUSER_BEGINNER_MTAG_BEGIN.....	81
2.2.8.1.1.3	TXUSER_BEGINNER_MTAG_BEGIN_LOG_FULL.....	82
2.2.8.1.1.4	TXUSER_BEGINNER_MTAG_BEGIN_NO_MEM.....	82
2.2.8.1.1.5	TXUSER_BEGINNER_MTAG_BEGUN.....	83
2.2.8.1.1.6	TXUSER_BEGINNER_MTAG_COMMIT	84
2.2.8.1.1.7	TXUSER_BEGINNER_MTAG_COMMIT_INDOUBT	85
2.2.8.1.1.8	TXUSER_BEGINNER_MTAG_COMMIT_TOO_LATE.....	86
2.2.8.1.1.9	TXUSER_BEGINNER_MTAG_REQUEST_COMPLETED	87
2.2.8.1.2	CONNTYPE_TXUSER_BEGIN2.....	87
2.2.8.1.2.1	TXUSER_BEGIN2_MTAG_ABORT	87
2.2.8.1.2.2	TXUSER_BEGIN2_MTAG_BEGIN.....	88
2.2.8.1.2.3	TXUSER_BEGIN2_MTAG_COMMIT	89
2.2.8.1.2.4	TXUSER_BEGIN2_MTAG_SINK_BEGUN	90
2.2.8.1.2.5	TXUSER_BEGIN2_MTAG_SINK_ERROR	91
2.2.8.1.2.6	TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE	92
2.2.8.1.2.7	TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT	93
2.2.8.1.2.8	TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE	94
2.2.8.1.3	CONNTYPE_TXUSER_PROMOTE.....	95

2.2.8.1.3.1	TXUSER_BEGINNER_MTAG_PROMOTE	95
2.2.8.2	Transaction Propagation	97
2.2.8.2.1	Pull Propagation.....	97
2.2.8.2.1.1	CONNTYPE_TXUSER_ASSOCIATE	97
2.2.8.2.1.1.1	TXUSER_ASSOCIATE_MTAG_ASSOCIATE	97
2.2.8.2.1.1.2	TXUSER_ASSOCIATE_MTAG_ASSOCIATED	99
2.2.8.2.1.1.3	TXUSER_ASSOCIATE_MTAG_COMM_FAILED	99
2.2.8.2.1.1.4	TXUSER_ASSOCIATE_MTAG_CREATE_BAD_TMADDR	100
2.2.8.2.1.1.5	TXUSER_ASSOCIATE_MTAG_LOG_FULL_LOCAL.....	101
2.2.8.2.1.1.6	TXUSER_ASSOCIATE_MTAG_LOG_FULL_REMOTE	101
2.2.8.2.1.1.7	TXUSER_ASSOCIATE_MTAG_NO_MEM_LOCAL.....	102
2.2.8.2.1.1.8	TXUSER_ASSOCIATE_MTAG_NO_MEM_REMOTE	103
2.2.8.2.1.1.9	TXUSER_ASSOCIATE_MTAG_TOO_LATE	103
2.2.8.2.1.1.10	TXUSER_ASSOCIATE_MTAG_TOO_MANY_LOCAL	104
2.2.8.2.1.1.11	TXUSER_ASSOCIATE_MTAG_TOO_MANY_REMOTE.....	105
2.2.8.2.1.1.12	TXUSER_ASSOCIATE_MTAG_TX_NOT_FOUND.....	105
2.2.8.2.2	Push Propagation	106
2.2.8.2.2.1	CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS	106
2.2.8.2.2.1.1	TXUSER_EXTENDEDWHEREABOUTS_MTAG_GET.....	106
2.2.8.2.2.1.2	TXUSER_EXTENDEDWHEREABOUTS_MTAG_GOT	107
2.2.8.2.2.1.3	TXUSER_EXTENDEDWHEREABOUTS_MTAG_NOMEM.....	108
2.2.8.2.2.2	CONNTYPE_TXUSER_EXPORT	109
2.2.8.2.2.2.1	TXUSER_EXPORT_MTAG_CREATE	109
2.2.8.2.2.2.2	TXUSER_EXPORT_MTAG_CREATE2	110
2.2.8.2.2.2.3	TXUSER_EXPORT_MTAG_CREATE_BAD_TMADDR.....	111
2.2.8.2.2.2.4	TXUSER_EXPORT_MTAG_CREATE_NET_TX_DISABLED	112
2.2.8.2.2.2.5	TXUSER_EXPORT_MTAG_CREATED.....	113
2.2.8.2.2.2.6	TXUSER_EXPORT_MTAG_EXPORT.....	113
2.2.8.2.2.2.7	TXUSER_EXPORT_MTAG_EXPORT_LOG_FULL.....	114
2.2.8.2.2.2.8	TXUSER_EXPORT_MTAG_EXPORT_NO_MEM.....	115
2.2.8.2.2.2.9	TXUSER_EXPORT_MTAG_EXPORT_TOO_LATE	116
2.2.8.2.2.2.10	TXUSER_EXPORT_MTAG_EXPORT_TOO_MANY	116
2.2.8.2.2.2.11	TXUSER_EXPORT_MTAG_EXPORT_TX_NOT_FOUND	117
2.2.8.2.2.2.12	TXUSER_EXPORT_MTAG_EXPORTED	118
2.2.8.2.2.3	CONNTYPE_TXUSER_IMPORT	118
2.2.8.2.2.3.1	TXUSER_IMPORT_MTAG_ABORT	118
2.2.8.2.2.3.2	TXUSER_IMPORT_MTAG_ABORT_TOO_LATE	119
2.2.8.2.2.3.3	TXUSER_IMPORT_MTAG_IMPORT	120
2.2.8.2.2.3.4	TXUSER_IMPORT_MTAG_IMPORT_TX_NOT_FOUND	121
2.2.8.2.2.3.5	TXUSER_IMPORT_MTAG_IMPORTED	122
2.2.8.2.2.3.6	TXUSER_IMPORT_MTAG_REQUEST_COMPLETED	123
2.2.8.2.2.4	CONNTYPE_TXUSER_IMPORT2	124
2.2.8.2.2.4.1	TXUSER_IMPORT2_MTAG_ABORT.....	124
2.2.8.2.2.4.2	TXUSER_IMPORT2_MTAG_IMPORT	125
2.2.8.2.2.4.3	TXUSER_IMPORT2_MTAG_IMPORT_WITH_SET	126
2.2.8.2.2.4.4	TXUSER_IMPORT2_MTAG_SINK_ERROR.....	128
2.2.8.2.2.4.5	TXUSER_IMPORT2_MTAG_SINK_IMPORTED.....	128
2.2.8.3	Transaction Administration	129
2.2.8.3.1	CONNTYPE_TXUSER_GETTXDETAILS	129
2.2.8.3.1.1	TXUSER_GETTXDETAILS_MTAG_GET.....	129
2.2.8.3.1.2	TXUSER_GETTXDETAILS_MTAG_GOTIT	130
2.2.8.3.1.3	TXUSER_GETTXDETAILS_MTAG_TX_NOT_FOUND	132
2.2.8.3.2	CONNTYPE_TXUSER_RESOLVE.....	133
2.2.8.3.2.1	TXUSER_RESOLVE_MTAG_ACCESSDENIED	133
2.2.8.3.2.2	TXUSER_RESOLVE_MTAG_CHILD_ABORT	134

2.2.8.3.2.3	TXUSER_RESOLVE_MTAG_CHILD_COMMIT	135
2.2.8.3.2.4	TXUSER_RESOLVE_MTAG_CHILD_NOT_PREPARED	136
2.2.8.3.2.5	TXUSER_RESOLVE_MTAG_FORGET_COMMITTED	137
2.2.8.3.2.6	TXUSER_RESOLVE_MTAG_FORGET_TX_NOT_COMMITTED	138
2.2.8.3.2.7	TXUSER_RESOLVE_MTAG_REQUEST_COMPLETE	139
2.2.8.3.2.8	TXUSER_RESOLVE_MTAG_TX_NOT_FOUND	140
2.2.8.3.3	CONNTYPE_TXUSER_SETTXTIMEOUT	140
2.2.8.3.3.1	TXUSER_SETTXTIMEOUT_MTAG_TX_NOT_FOUND	140
2.2.8.3.4	CONNTYPE_TXUSER_SETTXTIMEOUT2	141
2.2.8.3.5	CONNTYPE_TXUSER_TRACE	141
2.2.8.3.5.1	TXUSER_TRACE_MTAG_DUMP_TRANSACTION	141
2.2.8.3.5.2	TXUSER_TRACE_MTAG_REQUEST_COMPLETE	142
2.2.8.3.5.3	TXUSER_TRACE_MTAG_REQUEST_FAILED	143
2.2.8.3.5.4	TXUSER_TRACE_MTAG_TX_NOT_FOUND	144
2.2.8.4	Transaction Manager Administration	144
2.2.8.4.1	CONNTYPE_TXUSER_GETSECURITYFLAGS	144
2.2.8.4.1.1	TXUSER_GETSECURITYFLAGS_MTAG_FETCHED	144
2.2.8.4.1.2	TXUSER_GETSECURITYFLAGS_MTAG_GETSECURITYFLAGS	145
2.2.9	Connection Types Relevant to Transaction Managers	146
2.2.9.1	Transaction Propagation and Coordination	146
2.2.9.1.1	Push Propagation	146
2.2.9.1.1.1	CONNTYPE_PARTNERTM_PROPAGATE	146
2.2.9.1.1.1.1	PARTNERTM_PROPAGATE_MTAG_PROPAGATE	146
2.2.9.1.1.1.2	PARTNERTM_PROPAGATE_MTAG_PROPAGATED	148
2.2.9.1.1.1.3	PARTNERTM_PROPAGATE_MTAG_DUPLICATE	148
2.2.9.1.1.1.4	PARTNERTM_PROPAGATE_MTAG_NO_MEM	149
2.2.9.1.1.1.5	PARTNERTM_PROPAGATE_MTAG_LOG_FULL	150
2.2.9.1.1.1.6	PARTNERTM_PROPAGATE_MTAG_PREPAREREQ	150
2.2.9.1.1.1.7	PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE	151
2.2.9.1.1.1.8	PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR	152
2.2.9.1.1.1.9	PARTNERTM_PROPAGATE_MTAG_COMMITREQ	153
2.2.9.1.1.1.10	PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE	154
2.2.9.1.1.1.11	PARTNERTM_PROPAGATE_MTAG_ABORTREQ	154
2.2.9.1.1.1.12	PARTNERTM_PROPAGATE_MTAG_ABORTREQDONE	155
2.2.9.1.1.1.13	PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY	156
2.2.9.1.1.1.14	PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER	156
2.2.9.1.1.1.15	PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTERED	157
2.2.9.1.1.1.16	PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTRATIONREJECTED ..	158
2.2.9.1.1.1.17	PARTNERTM_PROPAGATE_MTAG_PHASE0	158
2.2.9.1.1.1.18	PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE	159
2.2.9.1.2	Pull Propagation	160
2.2.9.1.2.1	CONNTYPE_PARTNERTM_BRANCH	160
2.2.9.1.2.1.1	PARTNERTM_BRANCH_MTAG_BRANCH_LOG_FULL	160
2.2.9.1.2.1.2	PARTNERTM_BRANCH_MTAG_BRANCH_NO_MEM	161
2.2.9.1.2.1.3	PARTNERTM_BRANCH_MTAG_BRANCH_TOO_LATE	161
2.2.9.1.2.1.4	PARTNERTM_BRANCH_MTAG_BRANCH_TOO_MANY	162
2.2.9.1.2.1.5	PARTNERTM_BRANCH_MTAG_BRANCH_TX_NOT_FOUND	163
2.2.9.1.2.1.6	PARTNERTM_BRANCH_MTAG_BRANCHED	163
2.2.9.1.2.1.7	PARTNERTM_BRANCH_MTAG_BRANCHING	164
2.2.9.2	Transaction Recovery	165
2.2.9.2.1	Subordinate-Driven	165
2.2.9.2.1.1	CONNTYPE_PARTNERTM_CHECKABORT	165
2.2.9.2.1.1.1	PARTNERTM_CHECKABORT_MTAG_CHECK	165
2.2.9.2.1.1.2	PARTNERTM_CHECKABORT_MTAG_ABORTED	166
2.2.9.2.1.1.3	PARTNERTM_CHECKABORT_MTAG_RETRY	167

2.2.9.2.2	Superior-Driven	168
2.2.9.2.2.1	CONNTYPE_PARTNERTM_REDELIVERCOMMIT	168
2.2.9.2.2.1.1	PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQ.....	168
2.2.9.2.2.1.2	PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQDONE.....	169
2.2.9.2.2.1.3	PARTNERTM_REDELIVERCOMMIT_MTAG_RETRY	169
2.2.10	Connection Types Relevant to Resource Managers	170
2.2.10.1	Resource Manager Registration	170
2.2.10.1.1	CONNTYPE_TXUSER_RESOURCEMANAGER	170
2.2.10.1.1.1	TXUSER_RESOURCEMANAGER_MTAG_CREATE	170
2.2.10.1.1.2	TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE.....	172
2.2.10.1.1.3	TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE	172
2.2.10.1.1.4	TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE.....	173
2.2.10.1.2	CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL	174
2.2.10.1.2.1	TXUSER_RESOURCEMANAGERINTERNAL_MTAG_DUPLICATEDETECTED ...	174
2.2.10.2	Transaction Coordination	174
2.2.10.2.1	CONNTYPE_TXUSER_PHASE0	174
2.2.10.2.1.1	TXUSER_PHASE0_MTAG_CREATE	175
2.2.10.2.1.2	TXUSER_PHASE0_MTAG_CREATE_TOO_LATE	175
2.2.10.2.1.3	TXUSER_PHASE0_MTAG_CREATE_TX_NOT_FOUND	176
2.2.10.2.1.4	TXUSER_PHASE0_MTAG_CREATED	177
2.2.10.2.1.5	TXUSER_PHASE0_MTAG_PHASE0REQ.....	177
2.2.10.2.1.6	TXUSER_PHASE0_MTAG_PHASE0REQ_ABORT	178
2.2.10.2.1.7	TXUSER_PHASE0_MTAG_PHASE0REQDONE	179
2.2.10.2.1.8	TXUSER_PHASE0_MTAG_UNENLIST	179
2.2.10.2.2	CONNTYPE_TXUSER_ENLISTMENT.....	180
2.2.10.2.2.1	TXUSER_ENLISTMENT_MTAG_ABORTREQ	180
2.2.10.2.2.2	TXUSER_ENLISTMENT_MTAG_ABORTREQDONE	181
2.2.10.2.2.3	TXUSER_ENLISTMENT_MTAG_COMMITREQ	182
2.2.10.2.2.4	TXUSER_ENLISTMENT_MTAG_COMMITREQDONE	182
2.2.10.2.2.5	TXUSER_ENLISTMENT_MTAG_ENLIST	183
2.2.10.2.2.6	TXUSER_ENLISTMENT_MTAG_ENLIST_LOG_FULL.....	184
2.2.10.2.2.7	TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_LATE	185
2.2.10.2.2.8	TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_MANY	186
2.2.10.2.2.9	TXUSER_ENLISTMENT_MTAG_ENLIST_TX_NOT_FOUND	186
2.2.10.2.2.10	TXUSER_ENLISTMENT_MTAG_ENLISTED	187
2.2.10.2.2.11	TXUSER_ENLISTMENT_MTAG_PREPAREREQ	188
2.2.10.2.2.12	TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE	188
2.2.10.3	Transaction Recovery	189
2.2.10.3.1	CONNTYPE_TXUSER_REENLIST.....	189
2.2.10.3.1.1	TXUSER_REENLIST_MTAG_REENLIST	190
2.2.10.3.1.2	TXUSER_REENLIST_MTAG_REENLIST_ABORTED	191
2.2.10.3.1.3	TXUSER_REENLIST_MTAG_REENLIST_COMMITTED	191
2.2.10.3.1.4	TXUSER_REENLIST_MTAG_REENLIST_TIMEOUT	192
2.2.10.4	Voting	193
2.2.10.4.1	CONNTYPE_TXUSER_VOTER	193
2.2.10.4.1.1	TXUSER_STATUS_MTAG_ABORTED	193
2.2.10.4.1.2	TXUSER_STATUS_MTAG_COMMITTED	193
2.2.10.4.1.3	TXUSER_STATUS_MTAG_INDOUBT	194
2.2.10.4.1.4	TXUSER_VOTER_MTAG_CREATE	195
2.2.10.4.1.5	TXUSER_VOTER_MTAG_CREATE_TOO_LATE.....	195
2.2.10.4.1.6	TXUSER_VOTER_MTAG_CREATE_TX_NOT_FOUND	196
2.2.10.4.1.7	TXUSER_VOTER_MTAG_CREATED	197
2.2.10.4.1.8	TXUSER_VOTER_MTAG_VOTEREQ.....	197
2.2.10.4.1.9	TXUSER_VOTER_MTAG_VOTEREQDONE.....	198

3	Protocol Details	200
3.1	Common Details	200
3.1.1	Abstract Data Model	200
3.1.1.1	Converting a Name Object to an OLETX TM_ADDR Structure.....	201
3.1.1.2	Converting an OLETX TM_ADDR Structure to a Name Object	201
3.1.2	Timers	201
3.1.3	Initialization	201
3.1.4	Protocol Versioning Details	202
3.1.4.1	Supporting a Protocol Version	202
3.1.4.2	Negotiating a Common Protocol Version	202
3.1.4.3	Using the Negotiated Protocol Version	202
3.1.5	Higher-Layer Triggered Events.....	203
3.1.6	Message Processing Events and Sequencing Rules	203
3.1.7	Timer Events.....	203
3.1.8	Other Local Events.....	204
3.1.8.1	Connection Disconnected.....	204
3.2	Core Transaction Manager Facet Details.....	204
3.2.1	Abstract Data Model	204
3.2.1.1	Versioning.....	207
3.2.1.2	Transaction Logging.....	207
3.2.1.3	Enlistment Objects	208
3.2.1.4	Transaction States.....	208
3.2.1.4.1	Idle	210
3.2.1.4.2	Active	211
3.2.1.4.3	Phase Zero	211
3.2.1.4.4	Phase Zero Complete	212
3.2.1.4.5	Voting	212
3.2.1.4.6	Voting Complete	212
3.2.1.4.7	Phase One	213
3.2.1.4.8	Phase One Complete	213
3.2.1.4.9	Single Phase Commit.....	213
3.2.1.4.10	Committing.....	213
3.2.1.4.11	Aborting	213
3.2.1.4.12	In Doubt.....	214
3.2.1.4.13	Failed to Notify	214
3.2.1.4.14	Ended	214
3.2.1.5	Transaction Manager Facets	214
3.2.1.6	Protocol Extension Objects.....	215
3.2.2	Timers	215
3.2.2.1	Transaction Time-out Timer	216
3.2.3	Initialization	216
3.2.3.1	Core Transaction Manager Facet Initialization	218
3.2.3.2	Transaction Object Initialization.....	219
3.2.4	Higher-Layer Triggered Events.....	219
3.2.4.1	Recovery Event.....	219
3.2.5	Message Processing Events and Sequencing Rules	220
3.2.6	Timer Events.....	220
3.2.6.1	Transaction Timeout Timer	220
3.2.7	Other Local Events.....	220
3.2.7.1	Associate Transaction	221
3.2.7.2	Begin Commit.....	221
3.2.7.3	Begin In Doubt	222
3.2.7.4	Begin Phase One	222
3.2.7.5	Begin Phase Zero	222
3.2.7.6	Begin Rollback	223

3.2.7.7	Begin Voting.....	223
3.2.7.8	Branch Transaction Failure.....	224
3.2.7.9	Branch Transaction Success.....	224
3.2.7.10	Create Phase Zero Enlistment	225
3.2.7.11	Create Subordinate Enlistment	226
3.2.7.12	Create Superior Enlistment	226
3.2.7.13	Create Transaction	227
3.2.7.14	Create Voter Enlistment	228
3.2.7.15	Enlistment Commit Complete	228
3.2.7.16	Enlistment Phase One Complete	229
3.2.7.17	Enlistment Phase Zero Complete	230
3.2.7.18	Enlistment Rollback Complete	232
3.2.7.19	Enlistment Unilaterally Aborted	232
3.2.7.20	Enlistment Vote Complete.....	233
3.2.7.21	Export Transaction	234
3.2.7.22	Forget Transaction.....	235
3.2.7.23	Notify Aborted	235
3.2.7.24	Notify Recovered Transaction Committed.....	236
3.2.7.25	Phase One Completed	236
3.2.7.26	Propagate Transaction Failure	237
3.2.7.27	Propagate Transaction Success.....	238
3.2.7.28	Register Phase Zero Failure	238
3.2.7.29	Register Phase Zero Success.....	239
3.2.7.30	Resolve Transaction.....	239
3.2.7.31	Set Transaction Attributes	240
3.2.7.32	Set Transaction Timeout.....	241
3.2.7.33	Request Transaction Outcome	241
3.2.7.34	Unenlist Phase Zero Enlistment	242
3.2.7.35	Voting Complete	242
3.3	Application Details.....	243
3.3.1	Abstract Data Model.....	243
3.3.1.1	CONNTYPE_TXUSER_BEGINNER Initiator States.....	244
3.3.1.1.1	Idle	245
3.3.1.1.2	Awaiting Begin Response	246
3.3.1.1.3	Processing Transaction	246
3.3.1.1.4	Awaiting Commit Response	246
3.3.1.1.5	Awaiting Abort Response.....	246
3.3.1.1.6	Ended	246
3.3.1.2	CONNTYPE_TXUSER_BEGIN2 Initiator States.....	246
3.3.1.2.1	Idle	247
3.3.1.2.2	Awaiting Begin Response	247
3.3.1.2.3	Processing Transaction	247
3.3.1.2.4	Awaiting Set Timeout Response	248
3.3.1.2.5	Awaiting Commit Response	248
3.3.1.2.6	Awaiting Abort Response.....	248
3.3.1.2.7	Ended	248
3.3.1.3	CONNTYPE_TXUSER_PROMOTE Initiator States.....	248
3.3.1.3.1	Idle	249
3.3.1.3.2	Awaiting Promote Response.....	249
3.3.1.3.3	Processing Transaction	250
3.3.1.3.4	Awaiting Timeout Response.....	250
3.3.1.3.5	Awaiting Commit Response	250
3.3.1.3.6	Awaiting Abort Response.....	250
3.3.1.3.7	Ended	250
3.3.1.4	CONNTYPE_TXUSER_ASSOCIATE Initiator States	250

3.3.1.4.1	Idle	251
3.3.1.4.2	Awaiting Associate Response	251
3.3.1.4.3	Active	251
3.3.1.4.4	Ended	251
3.3.1.5	CONNTYPE_TXUSER_EXTENDWHEREABOUTS Initiator States.....	252
3.3.1.5.1	Idle	252
3.3.1.5.2	Awaiting Get Response	253
3.3.1.5.3	Ended	253
3.3.1.6	CONNTYPE_TXUSER_IMPORT Initiator States	253
3.3.1.6.1	Idle	254
3.3.1.6.2	Awaiting Import Response.....	254
3.3.1.6.3	Transaction Import Successful	254
3.3.1.6.4	Awaiting Abort Response.....	255
3.3.1.6.5	Ended	255
3.3.1.7	CONNTYPE_TXUSER_IMPORT2 Initiator States.....	255
3.3.1.7.1	Idle	256
3.3.1.7.2	Awaiting Import Response.....	256
3.3.1.7.3	Transaction Import Successful	257
3.3.1.7.4	Awaiting Abort Response.....	257
3.3.1.7.5	Ended	257
3.3.1.8	CONNTYPE_TXUSER_EXPORT Initiator States	257
3.3.1.8.1	Idle	258
3.3.1.8.2	Awaiting Create Response	258
3.3.1.8.3	Connection Active	259
3.3.1.8.4	Awaiting Export Response	259
3.3.1.8.5	Ended	259
3.3.1.9	CONNTYPE_TXUSER_GETTXDETAILS Initiator States	259
3.3.1.9.1	Idle	260
3.3.1.9.2	Awaiting Response	260
3.3.1.9.3	Ended	260
3.3.1.10	CONNTYPE_TXUSER_RESOLVE Initiator States	261
3.3.1.10.1	Idle	262
3.3.1.10.2	Awaiting Abort Response.....	263
3.3.1.10.3	Awaiting Forget Response	263
3.3.1.10.4	Awaiting Commit Response	263
3.3.1.10.5	Ended	263
3.3.1.11	CONNTYPE_TXUSER_SETTXTIMEOUT Initiator States.....	263
3.3.1.11.1	Idle	264
3.3.1.11.2	Awaiting Set Timeout Response	264
3.3.1.11.3	Ended	264
3.3.1.12	CONNTYPE_TXUSER_SETTXTIMEOUT2 Initiator States	265
3.3.1.12.1	Idle	265
3.3.1.12.2	Awaiting Set Timeout Response	265
3.3.1.12.3	Ended	266
3.3.1.13	CONNTYPE_TXUSER_TRACE Initiator States	266
3.3.1.13.1	Idle	266
3.3.1.13.2	Awaiting Trace Response	266
3.3.1.13.3	Ended	267
3.3.1.14	CONNTYPE_TXUSER_GETSECURITYFLAGS Initiator States	267
3.3.1.14.1	Idle	268
3.3.1.14.2	Awaiting Get Response	268
3.3.1.14.3	Ended	268
3.3.2	Timers	268
3.3.3	Initialization.....	268
3.3.4	Higher-Layer Triggered Events.....	268

3.3.4.1	Beginning a Transaction	268
3.3.4.1.1	Beginning a Transaction Using CONNTYPE_TXUSER_BEGIN2	269
3.3.4.1.2	Beginning a Transaction Using CONNTYPE_TXUSER_BEGINNER	269
3.3.4.1.3	Beginning a Transaction Using CONNTYPE_TXUSER_PROMOTE	270
3.3.4.2	Changing a Transaction Timeout.....	270
3.3.4.2.1	Changing a Transaction Timeout Using CONNTYPE_TXUSER_SETTXTIMEOUT	270
3.3.4.2.2	Changing a Transaction Timeout Using CONNTYPE_TXUSER_SETTXTIMEOUT2	271
3.3.4.3	Creating an Export Connection	272
3.3.4.4	Generating Trace Records for a Transaction Using CONNTYPE_TXUSER_TRACE....	273
3.3.4.5	Importing a Transaction	273
3.3.4.5.1	Importing a Transaction Using CONNTYPE_TXUSER_IMPORT	273
3.3.4.5.2	Importing a Transaction Using CONNTYPE_TXUSER_IMPORT2	274
3.3.4.6	Importing a Transaction with Additional Transaction Attributes.....	274
3.3.4.7	Initiating Transaction Commit	274
3.3.4.7.1	Commit a Transaction Using CONNTYPE_TXUSER_BEGIN2	275
3.3.4.7.2	Commit a Transaction Using CONNTYPE_TXUSER_BEGINNER	275
3.3.4.7.3	Commit a Transaction Using CONNTYPE_TXUSER_PROMOTE	276
3.3.4.8	Initiating Transaction Rollback.....	276
3.3.4.8.1	Abort a Transaction Using CONNTYPE_TXUSER_BEGIN2	276
3.3.4.8.2	Abort a Transaction Using CONNTYPE_TXUSER_BEGINNER	277
3.3.4.8.3	Abort a Transaction Using CONNTYPE_TXUSER_IMPORT	277
3.3.4.8.4	Abort a Transaction Using CONNTYPE_TXUSER_IMPORT2	277
3.3.4.8.5	Roll Back a Transaction Using CONNTYPE_TXUSER_PROMOTE	278
3.3.4.9	Obtaining Extended Whereabouts Using CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS	278
3.3.4.10	Obtaining the Security Configuration of the Transaction Manager Using CONNTYPE_TXUSER_GETSECURITYFLAGS	278
3.3.4.10.1	Obtaining the Details for a Transaction.....	279
3.3.4.11	Pulling a Transaction.....	279
3.3.4.12	Push a Transaction Using an Existing Export Connection	280
3.3.4.13	Resolving a Transaction.....	280
3.3.5	Message Processing Events and Sequencing Rules	281
3.3.5.1	Transaction Initiation and Completion.....	281
3.3.5.1.1	CONNTYPE_TXUSER_BEGINNER as Initiator	281
3.3.5.1.1.1	Receiving a TXUSER_BEGINNER_MTAG_BEGUN Message	281
3.3.5.1.1.2	Receiving a TXUSER_BEGINNER_MTAG_BEGIN_NO_MEM or TXUSER_BEGINNER_MTAG_BEGIN_LOG_FULL Message	282
3.3.5.1.1.3	Receiving a TXUSER_BEGINNER_MTAG_REQUEST_COMPLETED Message.....	282
3.3.5.1.1.4	Receiving a TXUSER_BEGINNER_MTAG_COMMIT_TOO_LATE Message	282
3.3.5.1.1.5	Receiving a TXUSER_BEGINNER_MTAG_COMMIT_INDOUBT Message	282
3.3.5.1.1.6	Connection Disconnected.....	283
3.3.5.1.2	CONNTYPE_TXUSER_BEGIN2 as Initiator.....	283
3.3.5.1.2.1	Receiving a TXUSER_BEGIN2_MTAG_SINK_BEGUN Message.....	283
3.3.5.1.2.2	Receiving a TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE Message.....	283
3.3.5.1.2.3	Receiving a TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE Message	284
3.3.5.1.2.4	Receiving a TXUSER_BEGIN2_MTAG_SINK_ERROR Message.....	284
3.3.5.1.2.5	Connection Disconnected.....	285
3.3.5.1.3	CONNTYPE_TXUSER_PROMOTE as Initiator	285
3.3.5.2	Transaction Propagation	285
3.3.5.2.1	Pull Propagation.....	285
3.3.5.2.1.1	CONNTYPE_TXUSER_ASSOCIATE as Initiator	285
3.3.5.2.1.1.1	Receiving a TXUSER_ASSOCIATE_MTAG_ASSOCIATED Message	285

3.3.5.2.1.1.2	Receiving Other TXUSER_ASSOCIATE_MTAG Messages	286
3.3.5.2.1.1.3	Receiving a TXUSER_IMPORT2_MTAG_SINK_ERROR Message	286
3.3.5.2.1.1.4	Connection Disconnected	287
3.3.5.2.2	Push Propagation	287
3.3.5.2.2.1	CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS as Initiator	287
3.3.5.2.2.1.1	Receiving a TXUSER_EXTENDEDWHEREABOUTS_MTAG_GOT Message	287
3.3.5.2.2.1.2	Receiving a TXUSER_EXTENDEDWHEREABOUTS_MTAG_NOMEM Message	287
3.3.5.2.2.1.3	CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS Connection Disconnected.....	287
3.3.5.2.2.2	CONNTYPE_TXUSER_EXPORT as Initiator	288
3.3.5.2.2.2.1	Receiving a TXUSER_EXPORT_MTAG_CREATED Message	288
3.3.5.2.2.2.2	Receiving a TXUSER_EXPORT_MTAG_CREATE_BAD_TMADDR or TXUSER_EXPORT_MTAG_CREATE_NET_TX_DISABLED Message.....	288
3.3.5.2.2.2.3	Receiving a TXUSER_EXPORT_MTAG_EXPORTED Message	288
3.3.5.2.2.2.4	Receiving a TXUSER_EXPORT_MTAG_EXPORT_LOG_FULL, TXUSER_EXPORT_MTAG_EXPORT_NO_MEM, TXUSER_EXPORT_MTAG_EXPORT_TOO_LATE, TXUSER_EXPORT_MTAG_EXPORT_TOO_MANY, or TXUSER_EXPORT_MTAG_EXPORT_NOT_FOUND Message.....	288
3.3.5.2.2.2.5	CONNTYPE_TXUSER_EXPORT Connection Disconnected	289
3.3.5.2.2.3	CONNTYPE_TXUSER_IMPORT as Initiator	289
3.3.5.2.2.3.1	Receiving a TXUSER_IMPORT_MTAG_IMPORTED Message.....	289
3.3.5.2.2.3.2	Receiving a TXUSER_IMPORT_MTAG_IMPORT_TX_NOT_FOUND OR TXUSER_IMPORT_MTAG_ABORT_TOO_LATE Message	289
3.3.5.2.2.3.3	Receiving a TXUSER_IMPORT_MTAG_REQUEST_COMPLETED Message	290
3.3.5.2.2.3.4	Connection Disconnected	290
3.3.5.2.2.4	CONNTYPE_TXUSER_IMPORT2 as Initiator	290
3.3.5.2.2.4.1	Receiving a TXUSER_IMPORT2_MTAG_IMPORTED Message	290
3.3.5.2.2.4.2	Receiving a TXUSER_IMPORT2_MTAG_SINK_ERROR Message	290
3.3.5.2.2.4.3	CONNTYPE_TXUSER_IMPORT2 Connection Disconnected	291
3.3.5.3	Transaction Administration	291
3.3.5.3.1	CONNTYPE_TXUSER_GETTXDETAILS as Initiator	291
3.3.5.3.1.1	Receiving a TXUSER_GETTXDETAILS_MTAG_GOTIT Message.....	292
3.3.5.3.1.2	Receiving a TXUSER_GETTXDETAILS_MTAG_TX_NOT_FOUND Message ..	292
3.3.5.3.1.3	CONNTYPE_TXUSER_GETTXDETAILS Connection Disconnected	292
3.3.5.3.2	CONNTYPE_TXUSER_RESOLVE as Initiator.....	292
3.3.5.3.2.1	Receiving a TXUSER_RESOLVE_MTAG_REQUEST_COMPLETE Message.....	292
3.3.5.3.2.2	Receiving a TXUSER_RESOLVE_MTAG_ACCESSDENIED or TXUSER_RESOLVE_MTAG_TX_NOT_FOUND Message.....	293
3.3.5.3.2.3	Receiving a TXUSER_RESOLVE_MTAG_CHILD_NOT_PREPARED Message ..	293
3.3.5.3.2.4	Receiving a TXUSER_RESOLVE_MTAG_FORGET_TX_NOT_COMMITTED Message.....	293
3.3.5.3.2.5	Connection Disconnected.....	293
3.3.5.3.3	CONNTYPE_TXUSER_SETTXTIMEOUT as Initiator	294
3.3.5.3.3.1	Receiving a TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE Message.....	294
3.3.5.3.3.2	Receiving a TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE or TXUSER_SETTXTIMEOUT_MTAG_TX_NOT_FOUND Message	294
3.3.5.3.3.3	Connection Disconnected.....	294
3.3.5.3.4	CONNTYPE_TXUSER_SETTXTIMEOUT2 as Initiator	295
3.3.5.3.4.1	Receiving a TXUSER_SETTXTIMEOUT_MTAG_TX_NOT_FOUND Message ..	295
3.3.5.3.4.2	Connection Disconnected.....	295

3.3.5.3.5	CONNTYPE_TXUSER_TRACE as Initiator	296
3.3.5.3.5.1	Receiving a TXUSER_TRACE_MTAG_REQUEST_COMPLETE Message	296
3.3.5.3.5.2	Receiving a TXUSER_TRACE_MTAG_REQUEST_FAILED or TXUSER_TRACE_MTAG_TX_NOT_FOUND Message.....	296
3.3.5.3.5.3	Connection Disconnected.....	296
3.3.5.4	Transaction Manager Administration.....	296
3.3.5.4.1	CONNTYPE_TXUSER_GETSECURITYFLAGS as Initiator	296
3.3.5.4.1.1	Receiving a TXUSER_GETSECURITYFLAGS _MTAG_FETCHED Message....	296
3.3.5.4.1.2	CONNTYPE_TXUSER_GETSECURITYFLAGS Connection Disconnected	297
3.3.6	Timer Events.....	297
3.3.7	Other Local Events	297
3.4	Transaction Manager Communicating with Application Details	297
3.4.1	Abstract Data Model	297
3.4.1.1	CONNTYPE_TXUSER_BEGINNER Acceptor States	298
3.4.1.1.1	Idle	299
3.4.1.1.2	Beginning Transaction	300
3.4.1.1.3	Active	300
3.4.1.1.4	Aborting Transaction	300
3.4.1.1.5	Transaction Aborted	300
3.4.1.1.6	Committing Transaction	300
3.4.1.1.7	Ended	300
3.4.1.2	CONNTYPE_TXUSER_BEGIN2 Acceptor States	300
3.4.1.2.1	Idle	301
3.4.1.2.2	Beginning Transaction	301
3.4.1.2.3	Active	302
3.4.1.2.4	Modifying Timeout	302
3.4.1.2.5	Aborting Transaction	302
3.4.1.2.6	Committing Transaction	302
3.4.1.2.7	Ended	302
3.4.1.3	CONNTYPE_TXUSER_PROMOTE Acceptor States.....	302
3.4.1.3.1	Idle	303
3.4.1.3.2	Beginning Transaction	303
3.4.1.3.3	Active	303
3.4.1.3.4	Modifying Timeout	304
3.4.1.3.5	Aborting Transaction	304
3.4.1.3.6	Committing Transaction	304
3.4.1.3.7	Ended	304
3.4.1.4	CONNTYPE_TXUSER_ASSOCIATE Acceptor States	304
3.4.1.4.1	Idle	305
3.4.1.4.2	Processing Associate Request	305
3.4.1.4.3	Active	306
3.4.1.4.4	Ended	306
3.4.1.5	CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS Acceptor States	306
3.4.1.5.1	Idle	307
3.4.1.5.2	Processing Inquiry	307
3.4.1.5.3	Ended	307
3.4.1.6	CONNTYPE_TXUSER_IMPORT Acceptor States	307
3.4.1.6.1	Idle	309
3.4.1.6.2	Processing Import Request.....	309
3.4.1.6.3	Active	309
3.4.1.6.4	Too Late to Abort.....	309
3.4.1.6.5	Processing Abort Request.....	309
3.4.1.6.6	Ended	309
3.4.1.7	CONNTYPE_TXUSER_IMPORT2 Acceptor States.....	309
3.4.1.7.1	Idle	310

3.4.1.7.2	Processing Import Request	311
3.4.1.7.3	Active	311
3.4.1.7.4	Too Late to Abort	311
3.4.1.7.5	Processing Abort Request	311
3.4.1.7.6	Ended	311
3.4.1.8	CONNTYPE_TXUSER_EXPORT Acceptor States	311
3.4.1.8.1	Idle	312
3.4.1.8.2	Processing Connection Request	313
3.4.1.8.3	Connection Active	313
3.4.1.8.4	Processing Push Operation Request	313
3.4.1.8.5	Ended	313
3.4.1.9	CONNTYPE_TXUSER_GETTXDETAILS Acceptor States	313
3.4.1.9.1	Idle	314
3.4.1.9.2	Processing Inquiry	314
3.4.1.9.3	Ended	314
3.4.1.10	CONNTYPE_TXUSER_RESOLVE Acceptor States	314
3.4.1.10.1	Idle	315
3.4.1.10.2	Processing Abort Request	316
3.4.1.10.3	Processing Forget Request	316
3.4.1.10.4	Processing Commit Request	316
3.4.1.10.5	Ended	316
3.4.1.11	CONNTYPE_TXUSER_SETTXTIMEOUT Acceptor States	316
3.4.1.11.1	Idle	317
3.4.1.11.2	Processing Request	317
3.4.1.11.3	Ended	317
3.4.1.12	CONNTYPE_TXUSER_SETTXTIMEOUT2 Acceptor States	317
3.4.1.12.1	Idle	318
3.4.1.12.2	Processing Request	318
3.4.1.12.3	Ended	318
3.4.1.13	CONNTYPE_TXUSER_TRACE Acceptor States	318
3.4.1.13.1	Idle	319
3.4.1.13.2	Processing Trace Request	319
3.4.1.13.3	Ended	319
3.4.1.14	CONNTYPE_TXUSER_GETSECURITYFLAGS Acceptor States	320
3.4.1.14.1	Idle	320
3.4.1.14.2	Processing Request	321
3.4.1.14.3	Ended	321
3.4.2	Timers	321
3.4.3	Initialization	321
3.4.4	Higher-Layer Triggered Events	322
3.4.5	Message Processing Events and Sequencing Rules	322
3.4.5.1	Transaction Initiation and Completion	322
3.4.5.1.1	CONNTYPE_TXUSER_BEGINNER as Acceptor	322
3.4.5.1.1.1	Receiving a TXUSER_BEGINNER_MTAG_BEGIN Message	322
3.4.5.1.1.2	Receiving a TXUSER_BEGINNER_MTAG_COMMIT Message	323
3.4.5.1.1.3	Receiving a TXUSER_BEGINNER_MTAG_ABORT Message	323
3.4.5.1.1.4	Connection Disconnected	323
3.4.5.1.2	CONNTYPE_TXUSER_BEGIN2 as Acceptor	324
3.4.5.1.2.1	Receiving a TXUSER_BEGIN2_MTAG_BEGIN Message	324
3.4.5.1.2.2	Receiving a TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT Message	325
3.4.5.1.2.3	Receiving a TXUSER_BEGIN2_MTAG_COMMIT Message	325
3.4.5.1.2.4	Receiving a TXUSER_BEGIN2_MTAG_ABORT Message	325
3.4.5.1.2.5	Connection Disconnected	326
3.4.5.1.3	CONNTYPE_TXUSER_PROMOTE as Acceptor	326
3.4.5.1.3.1	Receiving a TXUSER_BEGINNER_MTAG_PROMOTE Message	326

3.4.5.1.3.2	Receiving a TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT, TXUSER_BEGIN2_MTAG_COMMIT, or TXUSER_BEGIN2_MTAG_ABORT Message.....	327
3.4.5.1.3.3	Connection Disconnected.....	327
3.4.5.2	Transaction Propagation	327
3.4.5.2.1	Pull Propagation.....	327
3.4.5.2.1.1	CONNTYPE_TXUSER_ASSOCIATE as Acceptor	327
3.4.5.2.1.1.1	Receiving a TXUSER_ASSOCIATE_MTAG_ASSOCIATE Message	327
3.4.5.2.1.1.2	Connection Disconnected	329
3.4.5.2.2	Push Propagation	329
3.4.5.2.2.1	CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS as Acceptor.....	329
3.4.5.2.2.1.1	Receiving a TXUSER_EXTENDEDWHEREABOUTS_MTAG_GET Message	329
3.4.5.2.2.1.2	Connection Disconnected	330
3.4.5.2.2.2	CONNTYPE_TXUSER_EXPORT as Acceptor	330
3.4.5.2.2.2.1	Receiving a TXUSER_EXPORT_MTAG_CREATE Message	330
3.4.5.2.2.2.2	Receiving a TXUSER_EXPORT_MTAG_CREATE2 Message.....	331
3.4.5.2.2.2.3	Receiving a TXUSER_EXPORT_MTAG_EXPORT Message	331
3.4.5.2.2.2.4	Connection Disconnected	332
3.4.5.2.2.3	CONNTYPE_TXUSER_IMPORT as Acceptor	332
3.4.5.2.2.3.1	Receiving a TXUSER_IMPORT_MTAG_IMPORT Message.....	332
3.4.5.2.2.3.2	Receiving a TXUSER_IMPORT_MTAG_ABORT Message	333
3.4.5.2.2.3.3	Connection Disconnected	333
3.4.5.2.2.4	CONNTYPE_TXUSER_IMPORT2 as Acceptor	333
3.4.5.2.2.4.1	Receiving a TXUSER_IMPORT2_MTAG_IMPORT Message.....	333
3.4.5.2.2.4.2	Receiving a TXUSER_IMPORT2_MTAG_IMPORT_WITH_SET Message..	334
3.4.5.2.2.4.3	Receiving a TXUSER_IMPORT2_MTAG_ABORT Message	335
3.4.5.2.2.4.4	Connection Disconnected	335
3.4.5.3	Transaction Administration	335
3.4.5.3.1	CONNTYPE_TXUSER_GETTXDETAILS as Acceptor	335
3.4.5.3.1.1	Receiving a TXUSER_GETTXDETAILS_MTAG_GET Message	335
3.4.5.3.1.2	Connection Disconnected.....	336
3.4.5.3.2	CONNTYPE_TXUSER_RESOLVE as Acceptor.....	336
3.4.5.3.2.1	Receiving a TXUSER_RESOLVE_MTAG_CHILD_ABORT Message	336
3.4.5.3.2.2	Receiving a TXUSER_RESOLVE_MTAG_CHILD_COMMIT Message	337
3.4.5.3.2.3	Receiving a TXUSER_RESOLVE_MTAG_FORGET_COMMITTED Message	338
3.4.5.3.2.4	Connection Disconnected.....	338
3.4.5.3.3	CONNTYPE_TXUSER_SETTXTIMEOUT as Acceptor	338
3.4.5.3.3.1	Receiving a TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT Message....	338
3.4.5.3.3.2	Connection Disconnected.....	339
3.4.5.3.4	CONNTYPE_TXUSER_SETTXTIMEOUT2 as Acceptor	339
3.4.5.3.4.1	Receiving a TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT Message....	339
3.4.5.3.4.2	Connection Disconnected.....	339
3.4.5.3.5	CONNTYPE_TXUSER_TRACE as Acceptor	339
3.4.5.3.5.1	Receiving a TXUSER_TRACE_MTAG_DUMP_TRANSACTION Message.....	340
3.4.5.3.5.2	Connection Disconnected.....	340
3.4.5.4	Transaction Manager Administration.....	340
3.4.5.4.1	CONNTYPE_TXUSER_GETSECURITYFLAGS as Acceptor	340
3.4.5.4.1.1	Receiving a TXUSER_GETSECURITYFLAGS_MTAG_GETSECURITYFLAGS Message.....	340
3.4.5.4.1.2	Connection Disconnected.....	342
3.4.6	Timer Events.....	342
3.4.7	Other Local Events.....	342
3.4.7.1	Associate Transaction Failure	342
3.4.7.2	Associate Transaction Success.....	343

3.4.7.3	Begin Commit	344
3.4.7.4	Begin In Doubt	344
3.4.7.5	Begin Rollback	344
3.4.7.6	Begin Voting	345
3.4.7.7	Create Transaction Failure	345
3.4.7.8	Create Transaction Success	346
3.4.7.9	Create Voter Enlistment Failure	346
3.4.7.10	Create Voter Enlistment Success	347
3.4.7.11	Export Transaction Failure	347
3.4.7.12	Export Transaction Success.....	348
3.4.7.13	Phase One Complete.....	348
3.4.7.14	Phase Zero Complete	350
3.4.7.15	Register Phase Zero.....	351
3.4.7.16	Resolve Transaction Complete.....	351
3.4.7.17	Rollback Complete.....	352
3.4.7.18	Set Transaction Attributes Failure	352
3.4.7.19	Set Transaction Attributes Success	353
3.4.7.20	Set Transaction Timeout Failure	353
3.4.7.21	Set Transaction Timeout Success.....	354
3.4.7.22	Unilaterally Aborted	354
3.5	Resource Manager Details	354
3.5.1	Abstract Data Model	354
3.5.1.1	CONNTYPE_TXUSER_RESOURCEMANAGER Initiator States	355
3.5.1.1.1	Idle	356
3.5.1.1.2	Awaiting Create Response	356
3.5.1.1.3	Recovering.....	357
3.5.1.1.4	Awaiting Completion Confirmation	357
3.5.1.1.5	Active	357
3.5.1.1.6	Ended	357
3.5.1.2	CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL Initiator States	357
3.5.1.2.1	Idle	358
3.5.1.2.2	Awaiting Create Response	358
3.5.1.2.3	Recovering.....	359
3.5.1.2.4	Awaiting Completion Confirmation	359
3.5.1.2.5	Active	359
3.5.1.2.6	Ended	359
3.5.1.3	CONNTYPE_TXUSER_PHASE0 Initiator States	359
3.5.1.3.1	Idle	360
3.5.1.3.2	Awaiting Create Response	361
3.5.1.3.3	Active	361
3.5.1.3.4	Processing Phase 0 Request	361
3.5.1.3.5	Ended	361
3.5.1.4	CONNTYPE_TXUSER_ENLISTMENT Initiator States	361
3.5.1.4.1	Idle	362
3.5.1.4.2	Awaiting Enlistment Response	363
3.5.1.4.3	Active	363
3.5.1.4.4	Single Phase Committing	363
3.5.1.4.5	Preparing for Transaction Commit	363
3.5.1.4.6	Finalizing Commit Operations	363
3.5.1.4.7	Ended	363
3.5.1.5	CONNTYPE_TXUSER_REENLIST Initiator States.....	363
3.5.1.5.1	Idle	364
3.5.1.5.2	Awaiting Reenlist Response	364
3.5.1.5.3	Ended	364
3.5.1.6	CONNTYPE_TXUSER_VOTER Initiator States	365

3.5.1.6.1	Idle	366
3.5.1.6.2	Awaiting Creation Response	366
3.5.1.6.3	Active	367
3.5.1.6.4	Performing Transaction Operations	367
3.5.1.6.5	Awaiting Outcome.....	367
3.5.1.6.6	Ended	367
3.5.2	Timers	367
3.5.3	Initialization.....	367
3.5.4	Higher-Layer Triggered Events.....	368
3.5.4.1	Canceling Enlistment as a Phase Zero Participant on a Specific Transaction	368
3.5.4.2	Enlisting as a Phase Zero Participant on a Specific Transaction	368
3.5.4.3	Enlisting on a Specific Transaction	368
3.5.4.4	Enlistment Abort Request Completed.....	369
3.5.4.5	Enlistment Commit Request Completed	370
3.5.4.6	Enlistment Prepare Request Completed	370
3.5.4.7	Enlistment Single-Phase Commit Request Completed.....	371
3.5.4.8	Phase Zero Request Completed	372
3.5.4.9	Registering as a Voter on a Specific Transaction.....	372
3.5.4.10	Registering with Transaction Manager	373
3.5.4.10.1	Registering with Transaction Manager Using CONNTYPE_TXUSER_RESOURCEMANAGER	373
3.5.4.10.2	Registering with Transaction Manager Using CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL	373
3.5.4.11	Voter Vote Request Completed	374
3.5.5	Message Processing Events and Sequencing Rules	374
3.5.5.1	Resource Manager Registration	374
3.5.5.1.1	CONNTYPE_TXUSER_RESOURCEMANAGER as Initiator	374
3.5.5.1.1.1	Receiving a TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE Message ..	374
3.5.5.1.1.2	Receiving a TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE Message.....	375
3.5.5.1.1.3	Connection Disconnected.....	375
3.5.5.1.2	CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL as Initiator.....	375
3.5.5.1.2.1	Receiving a TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE or TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE Message	375
3.5.5.1.2.2	Receiving a TXUSER_RESOURCEMANAGERINTERNAL_MTAG_DUPLICATEDETECTED Message.....	375
3.5.5.1.2.3	Connection Disconnected.....	376
3.5.5.2	Transaction Coordination	376
3.5.5.2.1	CONNTYPE_TXUSER_PHASE0 as Initiator	376
3.5.5.2.1.1	Receiving a TXUSER_PHASE0_MTAG_CREATED Message	376
3.5.5.2.1.2	Receiving a TXUSER_PHASE0_MTAG_CREATE_TX_NOT_FOUND or TXUSER_PHASE0_MTAG_CREATE_TOO_LATE Message	376
3.5.5.2.1.3	Receiving a TXUSER_PHASE0_MTAG_PHASE0REQ Message	376
3.5.5.2.1.4	Receiving a TXUSER_PHASE0_MTAG_PHASE0REQ_ABORT Message	377
3.5.5.2.1.5	Connection Disconnected.....	377
3.5.5.2.2	CONNTYPE_TXUSER_ENLISTMENT as Initiator.....	377
3.5.5.2.2.1	Receiving a TXUSER_ENLISTMENT_MTAG_ENLISTED Message	377
3.5.5.2.2.2	Receiving a TXUSER_ENLISTMENT_MTAG_ENLIST_TX_NOT_FOUND, TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_LATE, TXUSER_ENLISTMENT_MTAG_ENLIST_LOG_FULL, or TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_MANY Message	378
3.5.5.2.2.3	Receiving a TXUSER_ENLISTMENT_MTAG_PREPAREREQ Message	378
3.5.5.2.2.4	Receiving a TXUSER_ENLISTMENT_MTAG_COMMITREQ Message.....	378
3.5.5.2.2.5	Receiving a TXUSER_ENLISTMENT_MTAG_ABORTREQ Message.....	378

3.5.5.2.2.6	Connection Disconnected.....	379
3.5.5.3	Transaction Recovery.....	379
3.5.5.3.1	CONNTYPE_TXUSER_REENLIST as Initiator.....	379
3.5.5.3.1.1	Receiving a TXUSER_REENLIST_MTAG_COMMITTED Message.....	379
3.5.5.3.1.2	Receiving a TXUSER_REENLIST_MTAG_ABORTED Message.....	379
3.5.5.3.1.3	Receiving a TXUSER_REENLIST_MTAG_REENLIST_TIMEOUT Message.....	380
3.5.5.3.1.4	Connection Disconnected.....	380
3.5.5.4	Voting.....	380
3.5.5.4.1	CONNTYPE_TXUSER_VOTER as Initiator.....	380
3.5.5.4.1.1	Receiving a TXUSER_VOTER_MTAG_CREATED Message.....	380
3.5.5.4.1.2	Receiving a TXUSER_VOTER_MTAG_CREATE_TX_NOT_FOUND or TXUSER_VOTER_MTAG_CREATE_TOO_LATE Message.....	380
3.5.5.4.1.3	Receiving a TXUSER_VOTER_MTAG_VOTEREQ Message.....	380
3.5.5.4.1.4	Receiving a TXUSER_STATUS_MTAG_COMMITTED Message.....	381
3.5.5.4.1.5	Receiving a TXUSER_STATUS_MTAG_ABORTED Message.....	381
3.5.5.4.1.6	Receiving a TXUSER_STATUS_MTAG_INDOUBT Message.....	381
3.5.5.4.1.7	Connection Disconnected.....	381
3.5.6	Timer Events.....	382
3.5.7	Other Local Events.....	382
3.5.7.1	Recover Transaction.....	382
3.5.7.2	Recover Transactions.....	382
3.5.7.3	Reenlistment Complete.....	383
3.5.7.4	Transaction Manager Down.....	383
3.6	Transaction Manager Communicating with Resource Manager Facet Details.....	383
3.6.1	Abstract Data Model.....	383
3.6.1.1	CONNTYPE_TXUSER_RESOURCEMANAGER Acceptor States.....	384
3.6.1.1.1	Idle.....	384
3.6.1.1.2	Creating.....	384
3.6.1.1.3	Reenlisting.....	384
3.6.1.1.4	Active.....	384
3.6.1.1.5	Ended.....	385
3.6.1.1.6	State Diagram.....	385
3.6.1.2	CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL Acceptor States.....	385
3.6.1.2.1	Idle.....	386
3.6.1.2.2	Creating.....	386
3.6.1.2.3	Reenlisting.....	386
3.6.1.2.4	Active.....	386
3.6.1.2.5	Ended.....	386
3.6.1.2.6	State Diagram.....	386
3.6.1.3	CONNTYPE_TXUSER_PHASE0 Acceptor States.....	387
3.6.1.3.1	Idle.....	387
3.6.1.3.2	Processing Enlistment Request.....	388
3.6.1.3.3	Active.....	388
3.6.1.3.4	Awaiting Phase Zero Response.....	388
3.6.1.3.5	Ended.....	388
3.6.1.3.6	State Diagram.....	388
3.6.1.4	CONNTYPE_TXUSER_ENLISTMENT Acceptor States.....	389
3.6.1.4.1	Idle.....	390
3.6.1.4.2	Processing Enlistment Request.....	390
3.6.1.4.3	Active.....	390
3.6.1.4.4	Awaiting Single-Phase Commit Response.....	390
3.6.1.4.5	Awaiting Prepare Response.....	390
3.6.1.4.6	Awaiting Prepare Response Aborted.....	390
3.6.1.4.7	Prepared.....	391
3.6.1.4.8	Awaiting Commit Response.....	391

3.6.1.4.9	Awaiting Abort Response.....	391
3.6.1.4.10	Ended	391
3.6.1.4.11	State Diagram.....	391
3.6.1.5	CONNTYPE_TXUSER_REENLIST Acceptor States.....	393
3.6.1.5.1	Idle	393
3.6.1.5.2	Processing Reenlist Request	394
3.6.1.5.3	Ended	394
3.6.1.5.4	State Diagram.....	394
3.6.1.6	CONNTYPE_TXUSER_VOTER Acceptor States	395
3.6.1.6.1	Idle	395
3.6.1.6.2	Create Voter	395
3.6.1.6.3	Active	395
3.6.1.6.4	Awaiting Voter Response.....	395
3.6.1.6.5	Awaiting Outcome.....	395
3.6.1.6.6	Ended	396
3.6.1.6.7	State Diagram.....	396
3.6.2	Timers	397
3.6.2.1	Reenlist Time-Out Timer.....	397
3.6.3	Initialization	397
3.6.4	Higher-Layer Triggered Events.....	397
3.6.5	Message Processing Events and Sequencing Rules	398
3.6.5.1	Resource Manager Registration	398
3.6.5.1.1	CONNTYPE_TXUSER_RESOURCEMANAGER as Acceptor	398
3.6.5.1.1.1	Receiving a TXUSER_RESOURCEMANAGER_MTAG_CREATE Message	398
3.6.5.1.1.2	Receiving a TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE Message.....	398
3.6.5.1.1.3	Connection Disconnected.....	399
3.6.5.1.2	CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL as Acceptor.....	399
3.6.5.1.2.1	Receiving a TXUSER_RESOURCEMANAGER_MTAG_CREATE Message	399
3.6.5.1.2.2	Receiving a TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE Message.....	399
3.6.5.1.2.3	Connection Disconnected.....	400
3.6.5.2	Transaction Coordination.....	400
3.6.5.2.1	CONNTYPE_TXUSER_PHASE0 as Acceptor.....	400
3.6.5.2.1.1	Receiving a TXUSER_PHASE0_MTAG_CREATE Message	400
3.6.5.2.1.2	Receiving a TXUSER_PHASE0_MTAG_PHASE0REQDONE Message	401
3.6.5.2.1.3	Receiving a TXUSER_PHASE0_MTAG_UNENLIST Message	401
3.6.5.2.1.4	Connection Disconnected.....	401
3.6.5.2.2	CONNTYPE_TXUSER_ENLISTMENT as Acceptor.....	402
3.6.5.2.2.1	Receiving a TXUSER_ENLISTMENT_MTAG_ENLIST Message	402
3.6.5.2.2.2	Receiving a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE Message...	403
3.6.5.2.2.3	Receiving a TXUSER_ENLISTMENT_MTAG_COMMITREQDONE Message....	404
3.6.5.2.2.4	Receiving a TXUSER_ENLISTMENT_MTAG_ABORTREQDONE Message.....	404
3.6.5.2.2.5	Connection Disconnected.....	405
3.6.5.3	Transaction Recovery	405
3.6.5.3.1	CONNTYPE_TXUSER_REENLIST as Acceptor.....	405
3.6.5.3.1.1	Receiving a TXUSER_REENLIST_MTAG_REENLIST Message.....	405
3.6.5.3.1.2	Connection Disconnected.....	406
3.6.5.4	Voting	406
3.6.5.4.1	CONNTYPE_TXUSER_VOTER as Acceptor	406
3.6.5.4.1.1	Receiving a TXUSER_VOTER_MTAG_CREATE Message.....	407
3.6.5.4.1.2	Receiving a TXUSER_VOTER_MTAG_VOTERREQDONE Message	407
3.6.5.4.1.3	Connection Disconnected.....	408

3.6.6	Timer Events.....	408
3.6.6.1	Reenlist Timeout Timer	408
3.6.7	Other Local Events.....	408
3.6.7.1	Begin Commit.....	409
3.6.7.2	Begin In Doubt	409
3.6.7.3	Begin Phase One	410
3.6.7.4	Begin Phase Zero	410
3.6.7.5	Begin Rollback	410
3.6.7.6	Begin Voting.....	411
3.6.7.7	Create Phase Zero Enlistment Failure	412
3.6.7.8	Create Phase Zero Enlistment Success.....	412
3.6.7.9	Create Resource Manager	412
3.6.7.10	Create Subordinate Enlistment Failure.....	413
3.6.7.11	Create Subordinate Enlistment Success	414
3.6.7.12	Create Voter Enlistment Failure	414
3.6.7.13	Create Voter Enlistment Success	414
3.6.7.14	Phase Zero Aborted	415
3.6.7.15	Reenlist Complete	415
3.6.7.16	Resource Manager Down	415
3.7	Superior Transaction Manager Facet Details	416
3.7.1	Abstract Data Model	416
3.7.1.1	CONNTYPE_PARTNERTM_PROPAGATE Initiator States	416
3.7.1.1.1	Idle	416
3.7.1.1.2	Awaiting Propagation Response.....	416
3.7.1.1.3	Active	417
3.7.1.1.4	Awaiting Abort Response.....	417
3.7.1.1.5	Phase Zero Registration	417
3.7.1.1.6	Requesting Phase Zero	417
3.7.1.1.7	Phase Zero	417
3.7.1.1.8	Phase Zero Registration During Phase Zero.....	417
3.7.1.1.9	Phase Zero with Outstanding Registration.....	417
3.7.1.1.10	Awaiting Prepare Response	418
3.7.1.1.11	Prepared	418
3.7.1.1.12	Awaiting Commit Response	418
3.7.1.1.13	Ended	418
3.7.1.1.14	State Diagram.....	418
3.7.1.2	CONNTYPE_PARTNERTM_BRANCH Acceptor States	419
3.7.1.2.1	Idle	420
3.7.1.2.2	Branching	420
3.7.1.2.3	Active	420
3.7.1.2.4	Awaiting Abort Response.....	420
3.7.1.2.5	Phase Zero Registration	420
3.7.1.2.6	Requesting Phase Zero	420
3.7.1.2.7	Phase Zero	420
3.7.1.2.8	Phase Zero Registration During Phase Zero.....	421
3.7.1.2.9	Phase Zero with Outstanding Registration.....	421
3.7.1.2.10	Awaiting Prepare Response	421
3.7.1.2.11	Prepared	421
3.7.1.2.12	Awaiting Commit Response	421
3.7.1.2.13	Ended	421
3.7.1.2.14	State Diagram.....	421
3.7.1.3	CONNTYPE_PARTNERTM_REDELIVERCOMMIT Initiator States	422
3.7.1.3.1	Idle	423
3.7.1.3.2	Awaiting Confirmation	423
3.7.1.3.3	Waiting to Rerequest.....	423

3.7.1.3.4	Ended	423
3.7.1.3.5	State Diagram.....	423
3.7.1.4	CONNTYPE_PARTNERTM_CHECKABORT Acceptor States.....	424
3.7.1.4.1	Idle	424
3.7.1.4.2	Processing Abort Inquiry	424
3.7.1.4.3	Ended	424
3.7.1.4.4	State Diagram.....	425
3.7.2	Timers	425
3.7.2.1	Redeliver Commit Timer	425
3.7.3	Initialization.....	426
3.7.4	Higher-Layer Triggered Events.....	426
3.7.5	Message Processing Events and Sequencing Rules	426
3.7.5.1	Transaction Propagation and Coordination.....	426
3.7.5.1.1	Push Propagation	426
3.7.5.1.1.1	CONNTYPE_PARTNERTM_PROPAGATE as Initiator.....	426
3.7.5.1.1.1.1	Receiving a PARTNERTM_PROPAGATE_MTAG_PROPAGATED Message	427
3.7.5.1.1.1.2	Receiving a PARTNERTM_PROPAGATE_MTAG_DUPLICATE, PARTNERTM_PROPAGATE_MTAG_NO_MEM, or PARTNERTM_PROPAGATE_MTAG_LOG_FULL Message.....	427
3.7.5.1.1.1.3	Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER, PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE, PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE, PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE, PARTNERTM_PROPAGATE_MTAG_ABORTREQDONE, or PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY Message.....	428
3.7.5.1.1.1.4	Receiving a PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR Message	428
3.7.5.1.1.1.5	Connection Disconnected	428
3.7.5.1.2	Pull Propagation.....	428
3.7.5.1.2.1	CONNTYPE_PARTNERTM_BRANCH as Acceptor.....	428
3.7.5.1.2.1.1	Receiving a PARTNERTM_BRANCH_MTAG_BRANCHING Message	428
3.7.5.1.2.1.2	Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER Message	429
3.7.5.1.2.1.3	Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE Message	429
3.7.5.1.2.1.4	Receiving a PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY Message	430
3.7.5.1.2.1.5	Receiving a PARTNERTM_PROPAGATE_MTAG_ABORTREQDONE Message	430
3.7.5.1.2.1.6	Receiving a PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE Message	431
3.7.5.1.2.1.7	Receiving a PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE Message	431
3.7.5.1.2.1.8	Receiving a PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR Message	432
3.7.5.1.2.1.9	Connection Disconnected	432
3.7.5.2	Transaction Recovery	433
3.7.5.2.1	Subordinate-Driven Recovery	433
3.7.5.2.1.1	CONNTYPE_PARTNERTM_CHECKABORT as Acceptor	433
3.7.5.2.1.1.1	Receiving a PARTNERTM_CHECKABORT_MTAG_CHECK Message	433
3.7.5.2.1.1.2	Connection Disconnected	434
3.7.5.2.2	Superior-Driven Recovery	434
3.7.5.2.2.1	CONNTYPE_PARTNERTM_REDELIVERCOMMIT as Initiator	434

3.7.5.2.2.1.1	Receiving a PARTNERTIM_REDELIVERCOMMIT_MTAG_COMMITREQDONE Message	434
3.7.5.2.2.1.2	Receiving a PARTNERTIM_REDELIVERCOMMIT_MTAG_RETRY Message	434
3.7.5.2.2.1.3	Connection Disconnected	435
3.7.6	Timer Events.....	435
3.7.6.1	Redeliver Commit Timer	435
3.7.7	Other Local Events.....	435
3.7.7.1	Begin Commit.....	436
3.7.7.2	Begin Phase One	436
3.7.7.3	Begin Phase Zero	437
3.7.7.4	Begin Rollback	437
3.7.7.5	Create Phase Zero Enlistment Failure	437
3.7.7.6	Create Phase Zero Enlistment Success.....	438
3.7.7.7	Create Subordinate Enlistment Failure	438
3.7.7.8	Create Subordinate Enlistment Success	439
3.7.7.9	Phase Zero Aborted	439
3.7.7.10	Propagate Transaction	439
3.8	Subordinate Transaction Manager Facet Details	440
3.8.1	Abstract Data Model	440
3.8.1.1	CONNTYPE_PARTNERTM_PROPAGATE Acceptor States	440
3.8.1.1.1	Idle	440
3.8.1.1.2	Propagating	440
3.8.1.1.3	Active	440
3.8.1.1.4	Aborting	441
3.8.1.1.5	Awaiting Registration Response.....	441
3.8.1.1.6	Awaiting Phase Zero.....	441
3.8.1.1.7	Awaiting Phase Zero Outcome	441
3.8.1.1.8	Awaiting Registration Response During Phase Zero	441
3.8.1.1.9	Awaiting Phase Zero Outcome with Outstanding Registration.....	441
3.8.1.1.10	Preparing.....	442
3.8.1.1.11	Prepared	442
3.8.1.1.12	Committing.....	442
3.8.1.1.13	Ended	442
3.8.1.1.14	State Diagram.....	442
3.8.1.2	CONNTYPE_PARTNERTM_BRANCH Initiator States	443
3.8.1.2.1	Idle	444
3.8.1.2.2	Awaiting Branch Response	444
3.8.1.2.3	Active	444
3.8.1.2.4	Aborting	444
3.8.1.2.5	Awaiting Registration Response.....	444
3.8.1.2.6	Awaiting Phase Zero.....	444
3.8.1.2.7	Awaiting Phase Zero Outcome	445
3.8.1.2.8	Awaiting Registration Response During Phase Zero	445
3.8.1.2.9	Awaiting Phase Zero Outcome with Outstanding Registration.....	445
3.8.1.2.10	Preparing.....	445
3.8.1.2.11	Prepared	445
3.8.1.2.12	Committing.....	445
3.8.1.2.13	Ended	445
3.8.1.2.14	State Diagram.....	446
3.8.1.3	CONNTYPE_PARTNERTM_REDELIVERCOMMIT Acceptor States	447
3.8.1.3.1	Idle	447
3.8.1.3.2	Processing Commit Inquiry	447
3.8.1.3.3	Ended	447

3.8.1.3.4	State Diagram.....	447
3.8.1.4	CONNTYPE_PARTNERTM_CHECKABORT Initiator States.....	448
3.8.1.4.1	Idle	448
3.8.1.4.2	Awaiting Confirmation	448
3.8.1.4.3	Waiting to Re-Request.....	449
3.8.1.4.4	Ended	449
3.8.1.4.5	State Diagram.....	449
3.8.2	Timers	449
3.8.2.1	Check Abort Timer.....	450
3.8.3	Initialization.....	450
3.8.4	Higher-Layer Triggered Events.....	450
3.8.5	Message Processing Events and Sequencing Rules	450
3.8.5.1	Transaction Propagation and Coordination.....	450
3.8.5.1.1	Push Propagation	450
3.8.5.1.1.1	CONNTYPE_PARTNERTM_PROPAGATE as Acceptor.....	450
3.8.5.1.1.1.1	Receiving a PARTNERTM_PROPAGATE_MTAG_PROPAGATE Message ..	451
3.8.5.1.1.1.2	Receiving Other PARTNERTM_PROPAGATE_MTAG Messages	452
3.8.5.1.1.1.3	Receiving a PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR Message	452
3.8.5.1.1.1.4	CONNTYPE_PARTNERTM_PROPAGATE Connection Disconnected.....	452
3.8.5.1.2	Pull Propagation.....	452
3.8.5.1.2.1	CONNTYPE_PARTNERTM_BRANCH as Initiator.....	452
3.8.5.1.2.1.1	Receiving a PARTNERTM_BRANCH_MTAG_BRANCHED Message	453
3.8.5.1.2.1.2	Receiving a PARTNERTM_BRANCH_MTAG_BRANCH_LOG_FULL, PARTNERTM_BRANCH_MTAG_BRANCH_NO_MEM, PARTNERTM_BRANCH_MTAG_BRANCH_TOO_LATE, PARTNERTM_BRANCH_MTAG_BRANCH_TOO_MANY, or PARTNERTM_BRANCH_MTAG_BRANCH_TX_NOT_FOUND Message	453
3.8.5.1.2.1.3	Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTERED Message	454
3.8.5.1.2.1.4	Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTRATIONREJECTED Message	454
3.8.5.1.2.1.5	Receiving a PARTNERTM_PROPAGATE_MTAG_ABORTREQ Message....	455
3.8.5.1.2.1.6	Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0 Message	455
3.8.5.1.2.1.7	Receiving a PARTNERTM_PROPAGATE_MTAG_PREPAREREQ Message.....	455
3.8.5.1.2.1.8	Receiving a PARTNERTM_PROPAGATE_MTAG_COMMITREQ Message..	455
3.8.5.1.2.1.9	Receiving a PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR Message	456
3.8.5.1.2.1.10	WConnection Disconnected.....	456
3.8.5.2	Transaction Recovery	457
3.8.5.2.1	Subordinate-Driven Recovery	457
3.8.5.2.1.1	CONNTYPE_PARTNERTM_CHECKABORT as Initiator	457
3.8.5.2.1.1.1	Receiving a PARTNERTM_CHECKABORT_MTAG_ABORTED Message ...	457
3.8.5.2.1.1.2	Receiving a PARTNERTM_CHECKABORT_MTAG_RETRY Message.....	457
3.8.5.2.1.1.3	CONNTYPE_PARTNERTM_CHECKABORT Connection Disconnected	457
3.8.5.2.2	Superior-Driven Recovery	458
3.8.5.2.2.1	Receiving a CONNTYPE_PARTNERTM_REDELIVERCOMMIT as Acceptor....	458
3.8.5.2.2.1.1	Receiving a PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQ Message	458
3.8.5.2.2.1.2	Connection Disconnected	459
3.8.6	Timer Events.....	459
3.8.6.1	Check Abort Timer.....	459
3.8.7	Other Local Events.....	459
3.8.7.1	Branch Transaction.....	459

3.8.7.2	Cancel Check Abort	460
3.8.7.3	Commit Complete.....	460
3.8.7.4	Create Superior Enlistment Success	460
3.8.7.5	Create Superior Enlistment Failure.....	461
3.8.7.6	Phase Zero Complete	461
3.8.7.7	Phase One Complete.....	462
3.8.7.8	Recover In Doubt Transaction	463
3.8.7.9	Register Phase Zero.....	463
3.8.7.10	Rollback Complete.....	464
3.8.7.11	Unilaterally Aborted	464
4	Protocol Examples	465
4.1	Simple Transaction Scenario	465
4.1.1	Beginning a Transaction.....	465
4.1.2	Completing a Transaction.....	467
4.1.2.1	Committing the Transaction	467
4.2	Transaction Marshaling Scenario (Pull Propagation)	467
4.2.1	Marshaling the Transaction.....	468
4.2.2	Unmarshaling the Transaction	469
4.2.3	Branching the Transaction.....	471
4.3	Transaction Marshaling Scenario (Push Propagation)	473
4.3.1	Obtaining the Whereabouts of the Receiver's Transaction Manager.....	473
4.3.2	Exporting the Transaction	474
4.3.3	Propagating the Transaction	476
4.3.4	Importing the Transaction.....	478
4.4	Simple Enlistment Scenario	480
4.4.1	Registering with the Transaction Manager as a Resource Manager.....	480
4.4.2	Enlisting in an Existing Transaction.....	481
4.4.3	Responding to Enlistment Notifications.....	483
4.4.3.1	Responding to a Prepare Request Message.....	483
4.4.3.2	Responding to a Commit Request Message.....	484
4.5	Transaction Manager Two-Phase Commit Scenario	484
4.5.1	Phase One	485
4.5.1.1	Phase One - Subordinate Resource Managers	485
4.5.1.2	Phase One - Subordinate Transaction Manager Facets.....	486
4.5.1.3	Phase One - The Root Transaction Manager.....	487
4.5.2	Phase Two	488
4.5.2.1	Phase Two - Subordinate Resource Managers	488
4.5.2.2	Phase Two - Subordinate Transaction Manager Facets	489
4.5.2.3	Phase Two - The Root Transaction Manager.....	490
4.6	Resource Manager Recovery Scenario	490
4.6.1	Initializing the Recovery Process	490
4.6.2	Reenlisting in In-Doubt Transactions.....	490
4.6.3	Completing Recovery.....	492
5	Security	494
5.1	Security Considerations for Implementers	494
5.2	Index of Security Parameters	495
6	Appendix A: Windows Behavior	496
7	Index.....	501

1 Introduction

This document specifies a comprehensive **distributed transaction** processing protocol that is referred to in this document as **OleTx**.

The MSDTC Connection Manager: OleTx Transaction Protocol is a concrete manifestation of the **Two-Phase Commit** protocol, as defined in [\[GRAY\]](#), for coordinating the **work** of multiple parties in a distributed system. This document specifies the syntax and semantics of the protocol but does not attempt to provide a primer on **transaction** processing in general. For more information about the Two-Phase Commit protocol, see [\[GRAY\]](#).

The MSDTC Connection Manager: OleTx Transaction Protocol uses the [MSDTC Connection Manager: OleTx Transports Protocol Specification](#) and [MSDTC Connection Manager: OleTx Multiplexing Protocol Specification](#) protocols, as specified in [\[MS-CMPO\]](#) and [\[MS-CMP\]](#), as a transport layer. This protocol provides concrete mechanisms for beginning, propagating, and completing **atomic transactions**. It also provides mechanisms for coordinating agreement on a single atomic **outcome** for each transaction, and for reliably distributing that outcome to all **participants** in the transaction.

The MSDTC Connection Manager: OleTx Transaction Protocol is applicable to **application** scenarios where atomic transaction processing is a requirement. This protocol is usable in network topologies where the MSDTC Connection Manager: OleTx Transports Protocol Specification (specified in [\[MS-CMPO\]](#)) and MSDTC Connection Manager: OleTx Multiplexing Protocol Specification (specified in [\[MS-CMP\]](#)) protocols are a viable network transport for establishing long-lived **session** relationships between the participants in an atomic transaction.

1.1 Glossary

The following terms are defined in [\[MS-GLOS\]](#):

- Abort request**
- Acceptor**
- Application**
- Atomic Transaction**
- Client**
- Commit Request**
- Connection**
- Connection Type**
- Contact Identifier**
- Core Transaction Manager Facet**
- Endpoint**
- Globally Unique Identifier (GUID)**
- Initiator**
- NULL GUID**
- Protocol Extension**
- Recovery**
- Server**
- Session**
- Single-Phase Commit**
- Subordinate Transaction Manager**
- Superior Transaction Manager**
- Transaction**
- Transaction Identifier**
- Transaction Manager (TM)**
- Transaction Propagation**
- Two-Phase Commit**

Unicode

The following terms are specific to this document:

Abort Outcome: One of the **outcomes** of an **atomic transaction**. The **abort outcome** indicates that the **work** performed during the lifetime of the **transaction** is discarded after the **transaction** completes. An **abort outcome** is reached when at least one **transaction participant** does not agree to commit the **transaction**.

Active Phase: The time during the lifetime of an **atomic transaction** before the **commit request** when the various **participants** in the **transaction** (**applications** and **resource managers**) perform all their intended **work** operations inside the **transaction**.

Begin Request: The action that is performed by a **root application** in order to create a new **atomic transaction**.

Commit Outcome: One of the **outcomes** of an **atomic transaction**. The **commit outcome** indicates that the **work** performed during the lifetime of the **transaction** will be retained after the **transaction** has completed, as specified by the ACID properties. A **commit outcome** is reached when all **transaction participants** agree to commit the **transaction**.

Distributed Transaction: A **transaction** that updates data on two or more networked computer systems. **Distributed transactions** extend the benefits of **transactions** to **applications** that must update distributed data.

Durable Resource: A **resource** whose state is expected to be retained beyond the lifetime of a particular **resource manager connection**. **Durable resources** are managed by **durable resource managers**.

Durable Resource Manager: A **resource manager** that manages **resources** whose states are expected to be maintained beyond the lifetime of a particular **resource manager connection**.

Enlistment: The relationship between a **participant** and a **transaction manager** in an **atomic transaction**. The term typically refers to the relationship between a **resource manager** and its **transaction manager**, or between a Subordinate Transaction Manager Facet and its Superior Transaction Manager Facet.

Extended Whereabouts: The data that is provided by a **protocol extension** and that indicates its network **endpoint** location and other information that is relevant to the **protocol extension**.

Facet: A subsystem in a **transaction manager** that maintains its own per-transaction state and responds to intra-transaction manager events from other **facets**. A **facet** can also be responsible for communicating with other **participants** of a **transaction**.

In Doubt Outcome: One of the **outcomes** of an **atomic transaction**. The **In Doubt outcome** indicates that a **commit request** was issued by the **root application**, but the **transaction manager** cannot ascertain the actual commit or abort decision.

Message Tag (MTAG): A 4-byte integer value that describes the message type and its interpretation.

OleTx: Common term encompassing the [\[MS-CMP\]](#), [\[MS-CMPO\]](#), [\[MS-CMOM\]](#), [\[MS-DTCM\]](#), [\[MS-DTCO\]](#), and [\[MS-DTCLU\]](#) protocols.

Outcome: One of the three possible results (Commit, Abort, In Doubt) reachable at the end of a life cycle for an **atomic transaction**.

Participant: Any of the parties that are involved in an **atomic transaction** and that have a stake in the operations that are performed under the **transaction** or in the **outcome** of the **transaction**.

Phase One Enlistment: An **enlistment** that indicates that the **subordinate participant** participates in Phase One.

Phase One Participant: A **participant** in a **Phase One enlistment**.

Phase Two Enlistment: An **enlistment** that indicates that the **subordinate participant** participates in Phase Two.

Phase Zero Enlistment: An **enlistment** that indicates that the **subordinate participant** participates in Phase Zero.

Phase Zero Participant: A **participant** with a **Phase Zero enlistment**.

Phase Zero Wave: A discrete stage inside Phase Zero processing in which Phase Zero notifications are sent to all known **Phase Zero enlistments**. New **Phase Zero enlistments** that appear during a **Phase Zero wave** are processed during the next **Phase Zero wave**. The process is repeated until a **Phase Zero wave** is processed without the creation of new **Phase Zero enlistments**.

Presumed Abort: An optimization of the **Two-Phase Commit** Protocol in which a **transaction manager** omits persisting **transaction abort outcomes** to a durable store.

Resource: A logical entity or unit of data whose state changes in accord with the **outcome** of an **atomic transaction**. **Resources** are either durable or volatile.

Resource Manager (RM): The **participant** that is responsible for coordinating the state of a **resource** with the **outcome** of **atomic transactions**. For a specified **transaction**, a **resource manager** enlists with exactly one **transaction manager** to vote on that **transaction outcome** and to obtain the final **outcome**. A **resource manager** is either durable or volatile, depending on its **resource**.

Resource Manager Identifier: The **GUID** that uniquely identifies the **resource manager**.

Resource Manager Session Identifier: The **GUID** that uniquely identifies a particular **session** between the **resource manager** and a **transaction manager**.

Rollback: Synonymous with abort.

Root Application: The **application** that is responsible for beginning and completing an **atomic transaction**. The **root application** communicates with a **root transaction manager** in order to begin and complete **transactions**.

Root Transaction Manager: The specific **transaction manager** that processes both the **Begin Request** and the **Commit Request** for a specified **transaction**. A specified **transaction** has exactly one **root transaction manager**.

Subordinate Participant: A role that is taken by a **participant** that is responsible for voting on the **outcome** of an **atomic transaction**. For a specified **transaction**, the set of **subordinate participants** is the set of all **resource managers** and the set of all **subordinate transaction managers**.

Transaction Description: An implementation-specific string that is associated with an **atomic transaction** and is often used to provide human-readable information about the **transaction**. Description strings are typically provided by the higher-layer software.

Transaction Lifetime: The lifetime of an **atomic transaction**. The **transaction lifetime** extends from the time when the **root transaction manager** processes a **begin request** to the time when all **participants** have forgotten the **transaction**.

Transaction Marshaling: The act of serializing and deserializing the information that is needed to carry out a **transaction propagation** action on a specified **transaction**.

Transient Failure: Any event that could result in a loss of transport connectivity between **participants**, such as a software crash, a software restart, or a temporary problem with network **connections**.

Volatile Resource: A **resource** whose value is not expected to be retained beyond the lifetime of a particular **resource manager connection**.

Volatile Resource Manager: A **resource manager** that manages **volatile resources**. A **volatile resource manager** does not perform **recovery** operations.

Voter: A **participant** in an **atomic transaction** that contributes to the final **outcome** of the **transaction** but does not manage access to **durable resources** or require **recovery** services. A **voter** votes on the **outcome** of the **transaction**, but it is provided only with best-effort **outcome** notifications by the **transaction manager**. A **volatile resource manager** typically acts as a **voter**.

Voter Enlistment: An **enlistment** that indicates that the **voter** participates in Phase One.

Whereabouts: Data that indicates the network **endpoint** location and properties of a **transaction manager**.

Work: The set of state changes that are applied to **resources** inside an **atomic transaction**.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified in [\[RFC2119\]](#). All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, <http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624>, as an additional source.

[C193] The Open Group, "Distributed TP: The XA Specification", February 1992, <http://www.opengroup.org/publications/catalog/c193.htm>

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, <http://www.opengroup.org/public/pubs/catalog/c706.htm>

[ISO-8859-1] International Organization for Standardization, "Information Technology -- 8-Bit Single-Byte Coded Graphic Character Sets -- Part 1: Latin Alphabet No. 1", ISO/IEC 8859-1, 1998,

<http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=28245&ICS1=35&ICS2=40&ICS3=>

Note There is a charge to download the specification.

[MS-CMOM] Microsoft Corporation, "[MSDTC Connection Manager: OleTx Management Protocol Specification](#)", September 2007.

[MS-CMP] Microsoft Corporation, "[MSDTC Connection Manager: OleTx Multiplexing Protocol Specification](#)", July 2007.

[MS-CMPO] Microsoft Corporation, "[MSDTC Connection Manager: OleTx Transports Protocol Specification](#)", July 2007.

[MS-GLOS] Microsoft Corporation, "[Windows Protocols Master Glossary](#)", March 2007.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>

[RFC2371] Lyon, J., Evans, K., and Klein, J., "Transaction Internet Protocol Version 3.0", RFC 2371, July 1998, <http://www.ietf.org/rfc/rfc2371.txt>

[RFC4122] Leach, P., Mealling, M., and Salz, R., "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, July 2005, <http://www.ietf.org/rfc/rfc4122.txt>

1.2.2 Informative References

[GRAY] Gray, G. and A. Reuter, "Transaction Processing: Concepts and Techniques", San Mateo, CA: Morgan Kaufmann Publishers, 1993, ISBN: 1558601902.

[MS-COM] Microsoft Corporation, "[Component Object Model Plus \(COM+\) Protocol Specification](#)", March 2007.

[MSDN-ANSI] Microsoft Corporation, "Unicode and Character Sets", <http://msdn2.microsoft.com/en-us/library/ms776440.aspx>

[MS-DTCLU] Microsoft Corporation, "[MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension](#)", September 2007.

[MSDN-DTCTrace] Microsoft Corporation, "DTC Tracing", <http://msdn2.microsoft.com/en-gb/library/ms678891.aspx>

[MS-DTCM] Microsoft Corporation, "[MSDTC Connection Manager: OleTx Transaction Internet Protocol Specification](#)", August 2007.

[MSDN-EVENT] Microsoft Corporation, "Event Logging", <http://msdn2.microsoft.com/en-us/library/aa363652.aspx>

[MS-MQMP] Microsoft Corporation, "[Message Queuing \(MSMQ\): Queue Manager Client Protocol Specification](#)", August 2007.

[MS-MQRR] Microsoft Corporation, "[Message Queuing \(MSMQ\): Queue Manager Remote Read Protocol Specification](#)", June 2007.

1.3 Protocol Overview (Synopsis)

This section presents a brief overview of the following topics:

- The life cycle of a transaction, including the Two-Phase Commit protocol.
- The distinct roles that are played by participants in transaction processing.
- Transaction **recovery** details.
- **Transaction marshaling** and propagation details.

1.3.1 Transaction Lifetime

At a general level, a transaction consists of a set of operations that an application or a set of applications treat as an atomic unit. These applications typically use one or more **resource managers** to modify and store the state that is affected by the transaction. The applications and resource managers make use of **transaction managers** to obtain a set of services. These roles are described further in section [1.3.3](#).

The lifetime of a transaction begins when an application determines that it needs a new transaction. The application assumes the role of **root application** and issues a **Begin request** to the **root transaction manager**. When a new transaction is created, either the root application or the root transaction manager assigns it an identifier that is unique in both time and space.

After the transaction is created, it enters the **active phase**. In the active phase, applications and resource managers perform all their intended actions inside the transaction.

Resource managers that perform work inside an atomic transaction contact their transaction manager to enlist on the transaction. By enlisting on a transaction, the resource manager is agreeing to participate in the Two-Phase Commit Protocol.

Applications and resource managers often share a transaction with a participant that is not located in the same operating system process or execution context. In this case, the application marshals the transaction to the other participant over an implementation-specific communication mechanism. If the receiving participant does not share a transaction manager with the sending participant, a **transaction propagation** handshake occurs to coordinate the transaction managers at both the sender and receiver of the transaction. After the transaction is successfully marshaled and (if needed) propagated, the receiving participant can perform operations on the transaction with its own transaction manager and also marshal the transaction to further participants.

As transaction **enlistment** and propagation occurs, the collection of resource managers and transaction managers relate to each other in a hierarchy known as a transaction tree.

The following figure depicts the transaction tree.

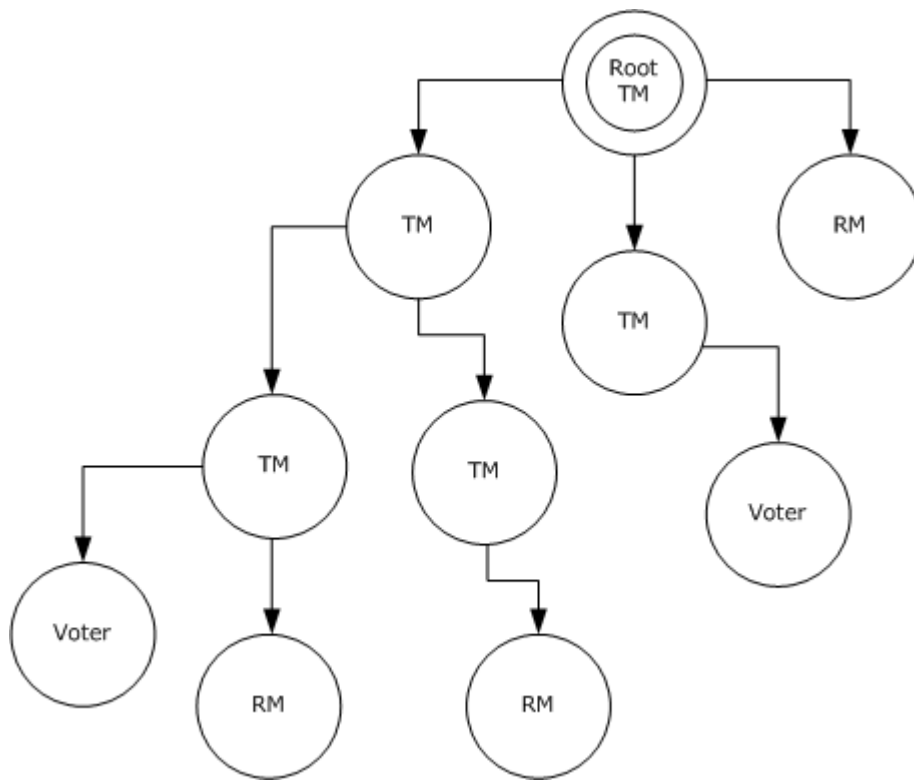


Figure 1: Transaction tree

Eventually, the root application that began the transaction determines that no more work is to be performed under the transaction. When that occurs, the application sends a **Commit request** to the root transaction manager to begin the process of completing the transaction.

When the root transaction manager receives the Commit request, it begins the process of determining the transaction outcome and communicating that outcome to all interested participants. That process begins with zero or more **Phase Zero waves** followed by [Phase One](#) and [Phase Two](#) of the Two-Phase Commit sequence.

1.3.1.1 Phase Zero

When a Commit request is issued by the root application, the transaction first enters Phase Zero. If there are no **Phase Zero participants**, the transaction leaves Phase Zero and proceeds to [Phase One](#).

Phase Zero is composed of one or more Phase Zero waves. At the beginning of a Phase Zero wave, all Phase Zero participants are notified that the transaction has entered Phase Zero. While the participants process the Phase Zero notification, they can continue to marshal the transaction to new participants. Consequently, participating transaction managers can still accept new enlistments during Phase Zero.

When a Phase Zero participant completes its Phase Zero processing, it sends a Phase Zero completion notification back to the transaction manager.

If any of the Phase Zero participants fail or issue an **Abort request** during the Phase Zero wave, the current Phase Zero wave is terminated and the transaction immediately moves to the aborting state, which is discussed in section [1.3.2.1](#).

Otherwise, after completion notifications are received from all Phase Zero participants:

- If no new **Phase Zero enlistments** were created during the current Phase Zero wave, the transaction proceeds to Phase One.
- If one or more new Phase Zero enlistments were created during the current Phase Zero wave, the transaction executes another Phase Zero wave with the new Phase Zero participants.

The following figure shows the overall Phase Zero flow.

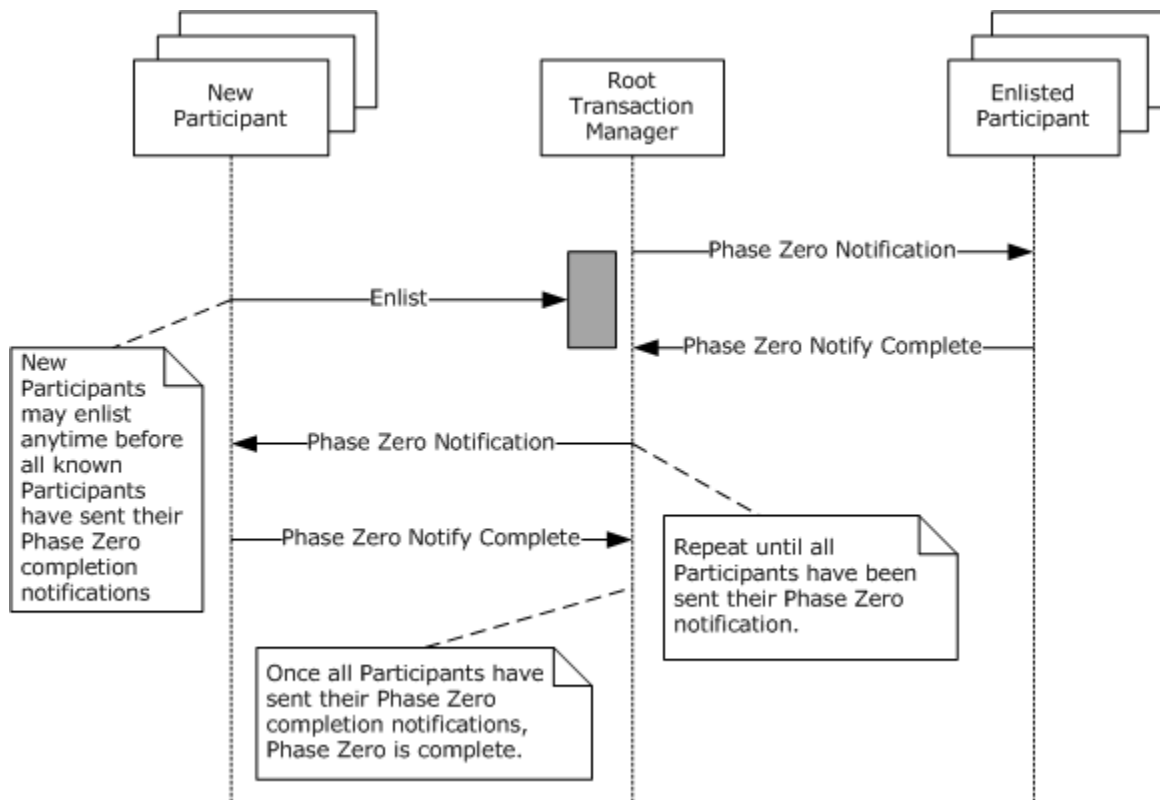


Figure 2: Transaction manager Phase Zero flow

1.3.1.2 Phase One

When Phase One begins, all transaction participants are now presumed to have completed their work inside the transaction.

During Phase One, each **Phase One participant** is asked to vote on the outcome of the transaction. Each participant vote can have one of three possible results:

- Read-Only: The participant agrees for the transaction to Commit but does not require an outcome notification.

- Prepared: The participant agrees for the transaction to Commit and requires an outcome notification.
- Aborted: The participant requires that the transaction abort.

Before a participant can vote Prepared, it performs whatever actions are necessary to be able to process an order to Commit or an order to Abort at some point in the future. Note that the request for a vote polls the transaction tree from the root transaction manager down to the leaf participants. When a **subordinate transaction manager** receives a request for a vote, it will first issue that request to all its immediate subordinates and process their votes before voting itself.

When all votes are collected by the root transaction manager, a decision about the transaction outcome is made. If every vote was either Read-Only or Prepared, the root transaction manager attempts to record a **Commit outcome** decision. If successful, the Commit outcome decision is officially made.

Otherwise, if one or more of the votes were Aborted or if a Commit outcome decision cannot be successfully recorded, the transaction manager makes an **Abort outcome** decision.

After an outcome decision is made, the root transaction manager proceeds to [Phase Two](#) in order to distribute outcome notification messages throughout the transaction tree.

The following figure depicts the Phase One flow.

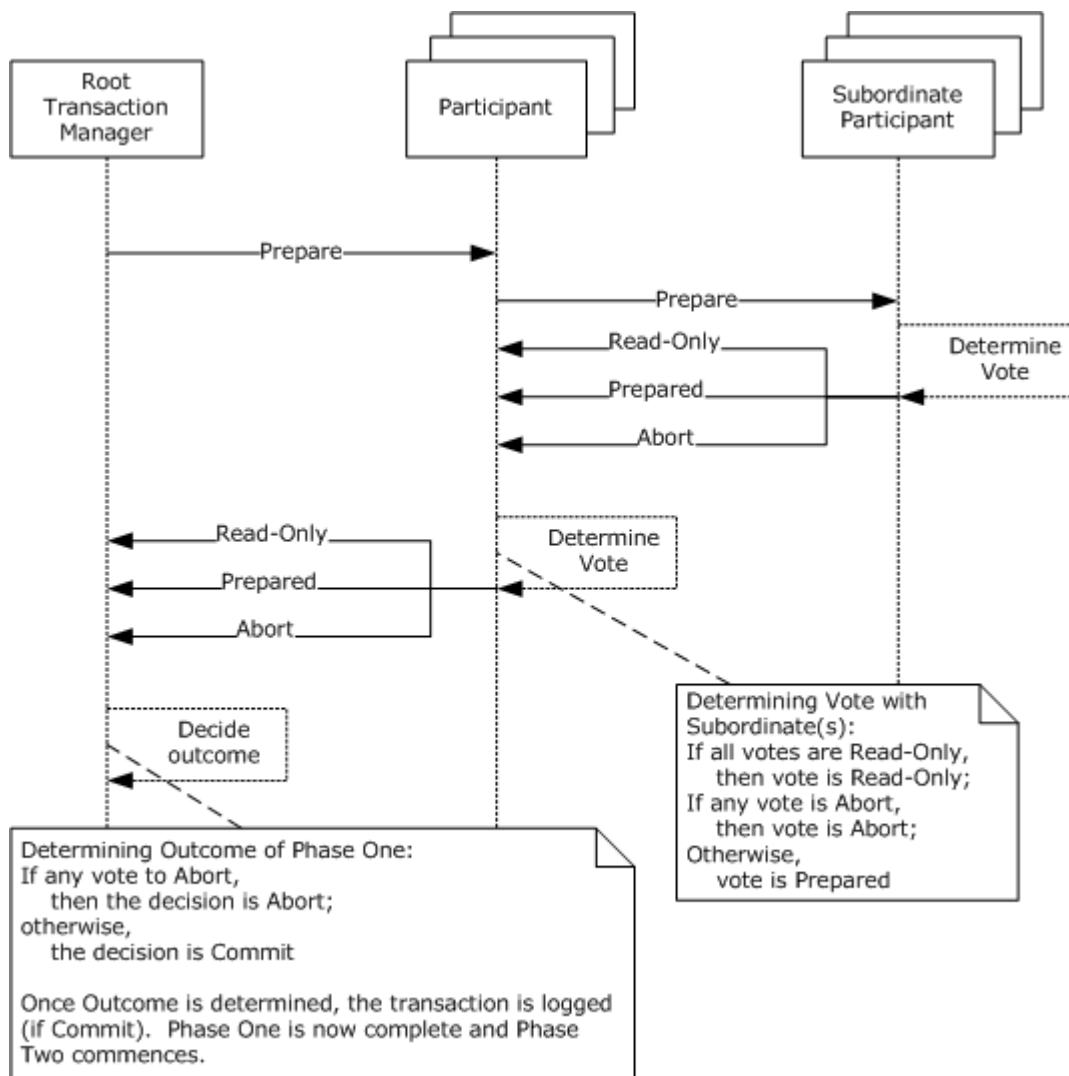


Figure 3: Transaction manager Phase One flow

1.3.1.3 Phase Two

When Phase Two begins, the root transaction manager has determined the transaction outcome.

If the transaction outcome is a Commit outcome, the transaction enters the committing state. Each participant that voted Prepared is sent an order to commit. The participants perform any necessary commit processing and respond with a committed notification.

If the transaction outcome is an Abort outcome, the transaction enters the [Aborting](#) state. Each participant that voted Prepared is sent an order to abort. The participants perform any necessary abort processing, and respond with an Aborted notification.

If a Prepared participant loses contact with its transaction manager, it is said to be [In Doubt](#). If it is a **durable resource manager** , it attempts to reconnect to the transaction manager and perform recovery in order to learn the outcome of the transaction. See section [1.3.4](#) for recovery details.

In general, participants (including the root application) are sent the outcome decision notification in parallel.

The following figure shows the Phase Two flow.

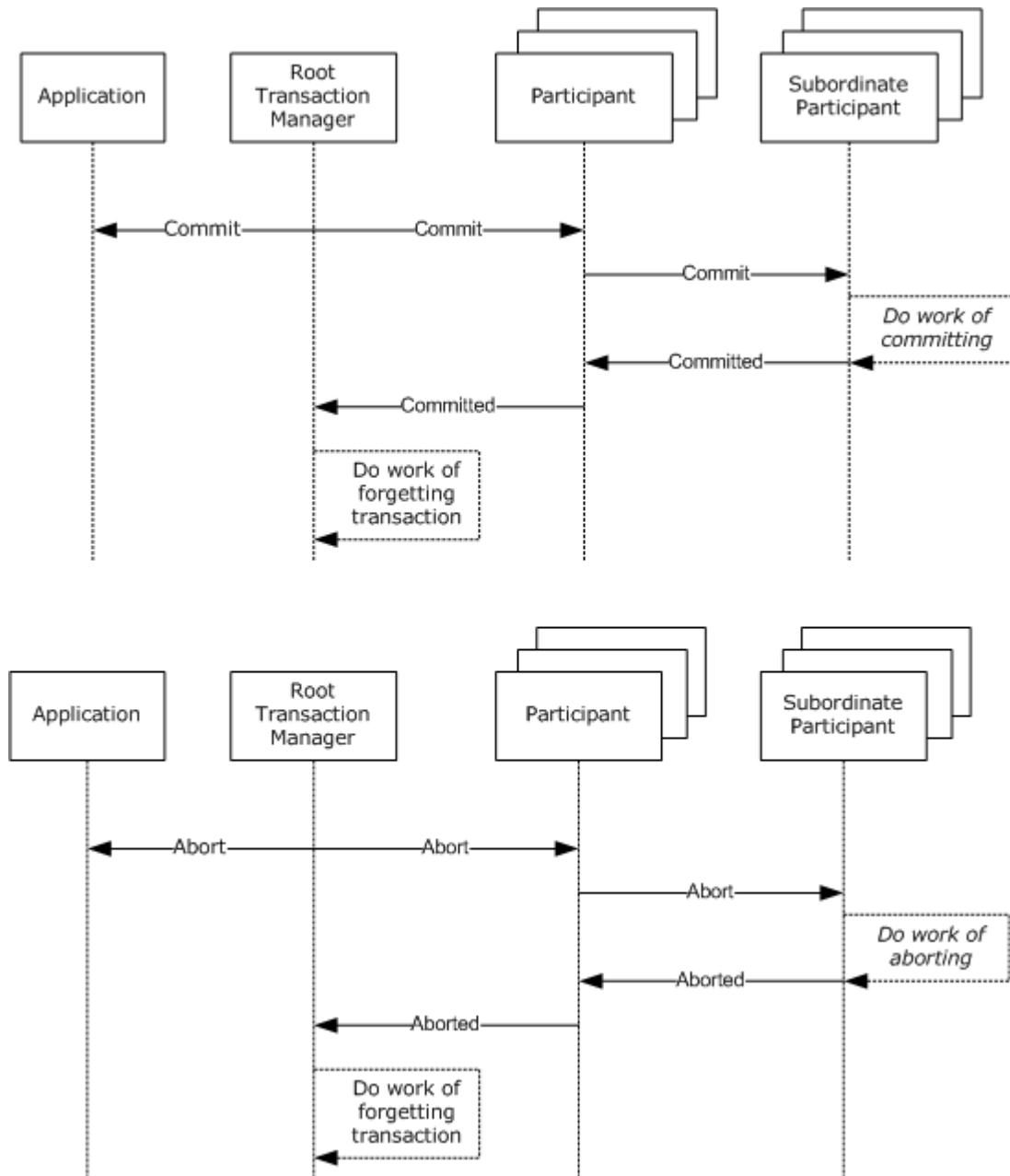


Figure 4: Transaction manager Phase Two flow

1.3.2 Additional Considerations

In addition to the two-phase commit processing described in the previous section, there are two more cases to consider:

- [Unilateral abort](#)
- [Single-phase commit](#)

1.3.2.1 Unilateral Abort

Until a participant votes on the outcome of the transaction, any participant can decide to unilaterally stop the transaction by issuing an Abort request to its transaction manager. This ability is known as a Unilateral Abort.

After a transaction manager receives an Abort request from one of its participants, it immediately transitions the transaction to the [Aborting](#) state, which guarantees an Abort outcome. All other participants will be notified of the Abort outcome although it is possible that the root application does not discover the Abort outcome until it attempts to complete the transaction or perform some other operation involving the transaction manager or another participant.

After a specified transaction manager enters the Aborting state, it does not issue any further [Phase Zero](#) notifications or [Phase One](#) requests to vote. For a transaction that spans two or more transaction managers due to propagation, it is possible for the Abort outcome decision to race with other Phase Zero or Phase One activity as it is communicated between the transaction managers.

The following figure shows the Unilateral Abort flow.

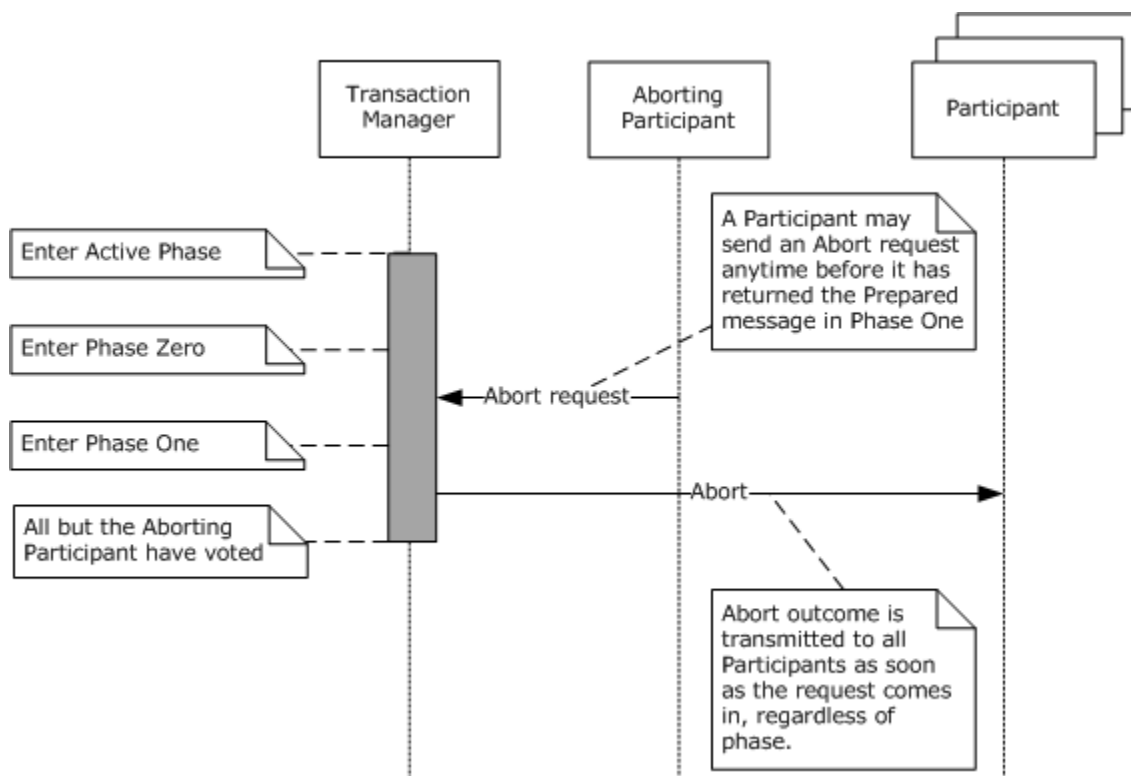


Figure 5: Unilateral Abort flow

1.3.2.2 Single-Phase Commit

If a transaction manager has exactly one subordinate **Phase One enlistment**, the transaction manager attempts to perform the **single-phase commit** optimization. In this case, the transaction manager sends the **subordinate participant** a request to perform a single-phase commit, instead of the standard [Phase One](#) Prepare request. This optimization delegates the right to decide the transaction outcome to the subordinate.

The subordinate accepts this delegation by making an outcome decision and eventually notifying the transaction manager; or it rejects the Single-Phase Commit request by responding Prepared. In the latter case, the transaction manager makes its own outcome decision and then engages in a standard [Phase Two](#) exchange with the participant.

There is a possible disadvantage to this optimization: if the transaction manager loses contact with the subordinate participant after sending the Single-Phase Commit request but before receiving an outcome notification, it has no reliable mechanism for recovering the actual outcome of the transaction. Consequently, the transaction manager sends an **In Doubt outcome** to any applications or **voters** awaiting informational outcome notification.

The single-phase commit optimization can be used by any transaction manager that has exactly one Phase One subordinate enlistment, not just the root transaction manager. For example, if transaction manager A only has transaction manager B as a subordinate enlistment, then A can use the single-phase commit optimization with B. If in the same transaction, B only has transaction manager C as a subordinate enlistment, it too can use the single-phase commit optimization with C. This is true regardless of the number of subordinate enlistments that are registered with C.

Note that a non-root transaction manager only performs the single-phase commit optimization if its own **superior transaction manager** has sent it a Single-Phase Commit request.

The following figure shows the Single-Phase Commit flow.

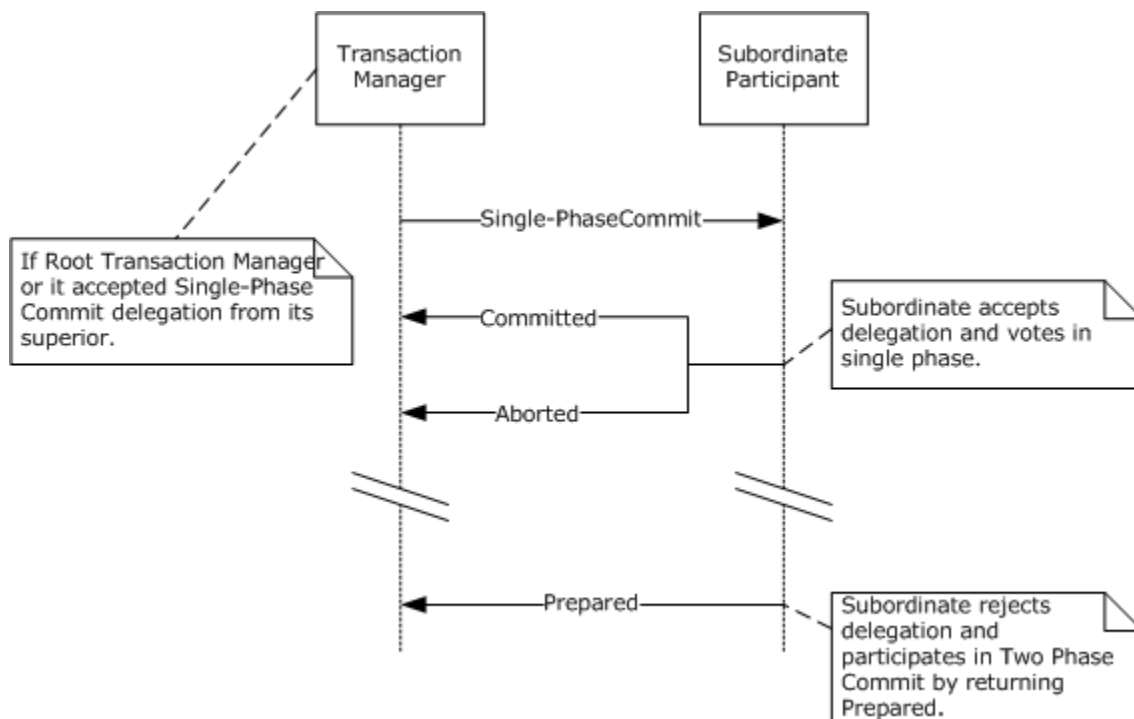


Figure 6: Single-Phase Commit flow

1.3.3 Transaction Roles

This protocol enables transaction processing to be distributed among two or more distinct participants. These participants are categorized according to three specialized roles that perform specific functions inside the transaction:

- [Application role](#)
- [Resource manager role](#)
- [Transaction manager role](#)

Each role is functionally independent of the other two. It is possible to implement the protocol functions that are required by any of these three roles without implementing the protocol functions that are required by the other two. For example, it is possible to implement a transactional resource manager without building a transaction manager or a transaction-aware application.

The following graphic depicts the transaction roles.

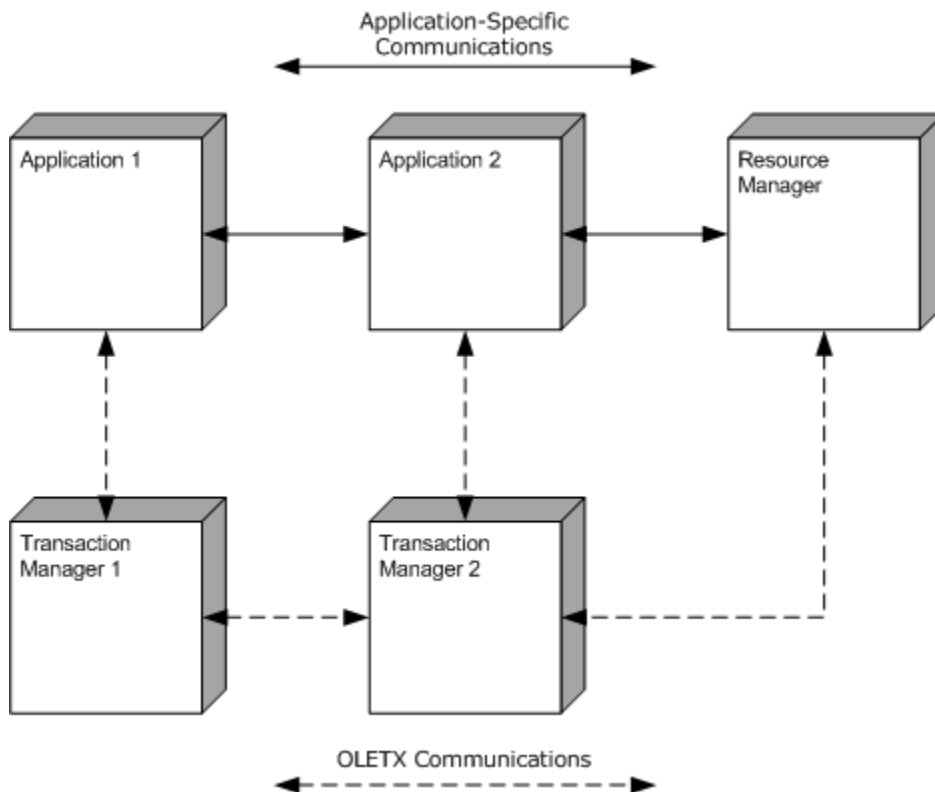


Figure 7: Transaction roles

1.3.3.1 Application Role

The application role is generally performed by user software programs that make use of transaction processing services in order to obtain greater reliability or reduce the complexity of error-handling tasks.

The application role is typically responsible for performing the following tasks:

- Determining when to begin a transaction.
- Marshaling the transaction to other applications and to resource managers.
- Propagating the transaction from one transaction manager to another.
- Determining when to complete a transaction.
- Performing administrative operations against a specific transaction.
- Performing administrative operations against the transaction manager.

In general, the motivations behind these tasks are implementation-specific. The protocol mechanisms by which these tasks can be accomplished are discussed in sections [2](#) and [3](#).

1.3.3.2 Resource Manager Role

The resource manager role is generally performed by software programs that manage transactional **resources**. Databases and queues are the most common examples of such programs.

This protocol supports three types of enlistments: Phase Zero enlistments, Phase One enlistments, and **voter enlistments**. These enlistment types correspond to three common categories of resource manager:

- Caching resource managers appear like a durable resource manager to an application, but they actually delegate their durable state changes to another resource manager that provides true durability. Caching resource managers typically use Phase Zero enlistments.
- Durable resource managers manage access to **durable resources**. They are expected to support recovery. Durable resource managers typically use Phase One enlistments.
- **Volatile resource managers** manage access to **volatile resources** whose state does not persist beyond the lifetime of the resource manager process. Volatile resource managers typically use voter enlistments.

The resource manager role is typically responsible for the following tasks:

- Providing applications with access to data in a transactional manner. This function is specific to the implementation of a resource manager.
- Registering with a transaction manager and performing recovery operations for all [In Doubt](#) transactions.
- Enlisting for various two-phase commit notifications.
- Voting on transaction outcomes in accord with the implementation-specific policies of the resource manager.

In general, the motivations behind these tasks are implementation-specific. The specific protocol mechanisms by which these tasks are accomplished are discussed in sections [2](#) and [3](#).

1.3.3.3 Transaction Manager Role

The Transaction Manager Role is generally performed by specialized middleware software programs that provide transactional services to applications and resource managers.

The transaction manager role is typically responsible for the following tasks:

- Providing the following services to applications and resource managers:
 - Beginning transactions.
 - Completing transactions.
 - Coordinating agreement with participants on the outcome of the transaction.
 - Reaching the decision to commit.
 - Ensuring the outcome decision is reliably distributed.
 - Coordinating the process of recovery if failures occur.
- Coordinating the outcome of individual transactions by using the Two-Phase Commit protocol.
- Coordinating recovery with other participants after a process or communication failure. See section [1.3.4](#) for recovery details.

A transaction manager is best understood as the aggregation of several cooperating software modules that work together to provide the services previously mentioned. This document calls these software modules **facets**, and assumes the presence of the following five facets:

- A facet that acts as a core transaction manager manager.
- A facet that communicates with applications.
- A facet that communicates with resource managers.
- A facet that acts as a superior transaction manager.
- A facet that acts as a subordinate transaction manager.

A transaction manager provides implementation-specific mechanisms to allow the facets to communicate with one another within the transaction manager itself.

In contrast, the transaction manager facets use the [MSDTC Connection Manager: OleTx Transports Protocol Specification](#) (specified in [MS-CMPO]) and [MSDTC Connection Manager: OleTx Multiplexing Protocol Specification](#) (specified in [MS-CMP]) protocols as transports for this protocol when they communicate with other participants (for example, applications, resource managers, and remote transaction managers). The subprotocols that are used to provide services to these participants are known as **connection types**. The specific connection types that are used in this protocol are specified in detail in section [3](#).

These facets are functionally dependent upon each other. A general purpose transaction manager is composed of all five of these facets.

The following figure shows the transaction manager facets.

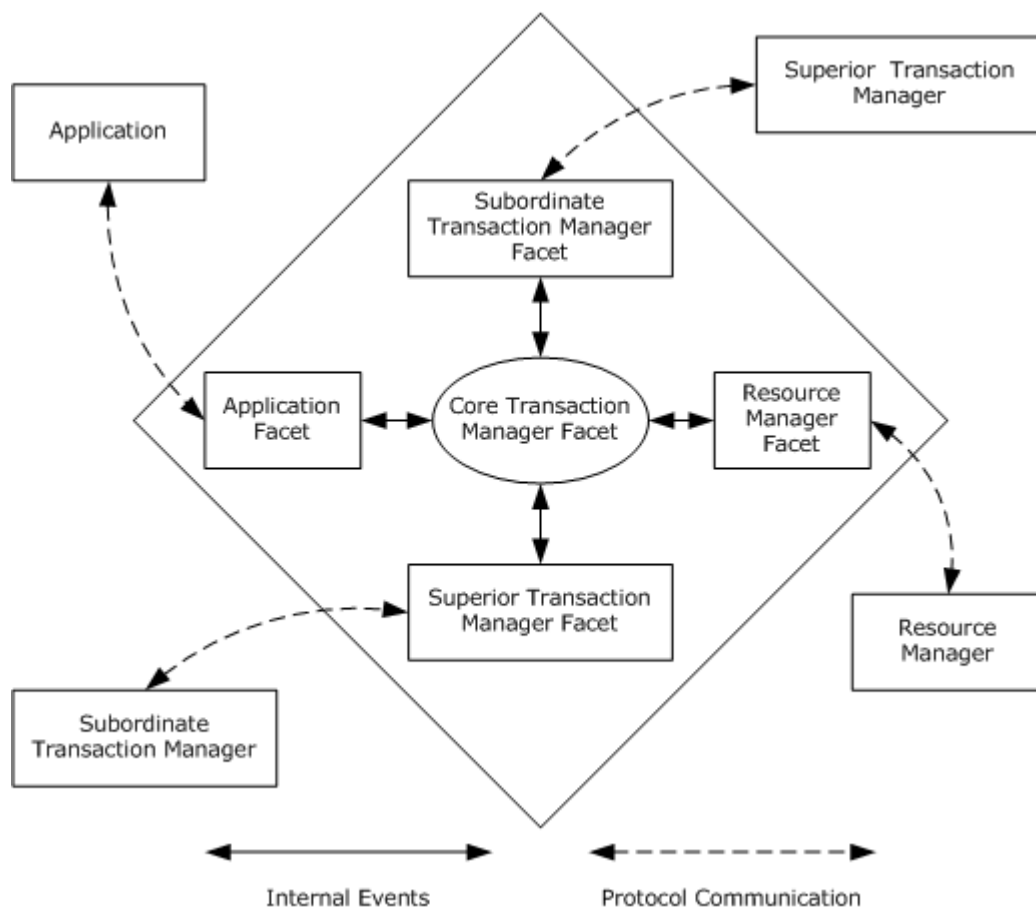


Figure 8: Transaction manager facets

1.3.3.3.1 Core Transaction Manager Facet

The **Core Transaction Manager Facet** is a logical construct in the context of this protocol. It never establishes network communication with any other transaction participant. It communicates with the other transaction manager facets through implementation-specific mechanisms.

1.3.3.3.2 Transaction Manager Communication with an Application Facet

When the transaction manager is communicating with an application facet, it provides the following services to applications:

- Transaction creation.
- Transaction propagation to a remote transaction manager.
- Transaction propagation by joining an existing transaction as a [subordinate transaction manager facet](#).
- Transaction completion.
- Administrative operations against a specific transaction. These operations include:

- Setting the time-out on a transaction.
- Obtaining transaction details, such as information about the [superior transaction manager facet](#) and the list of subordinate participants.
- Manually resolving the outcome of a transaction.
- Requesting that the transaction manager provide details of the transaction in its implementation-specific trace log.
- Administrative operations against the transaction manager. These operations include the ability to obtain information about the security configuration of the transaction manager.

1.3.3.3.3 Transaction Manager Communication with a Resource Manager Facet

When the transaction manager is communicating with a resource manager facet, it provides the following services to resource managers:

- Resource manager registration.
- Recovery and outcome notification for [In Doubt](#) transactions.
- Transaction enlistment for [Phase Zero](#), [Phase One](#), and voter participants.
- Phase Zero, Phase One, and [Phase Two](#) notifications inside the Two-Phase Commit protocol.

1.3.3.3.4 Superior Transaction Manager Facet

The Superior Transaction Manager Facet provides the following services to [subordinate transaction manager facets](#):

- Acts as Superior Transaction Manager Facet to a number of subordinate transaction manager facets in the Two-Phase Commit protocol.
- Provides recovery and outcome notification for transactions that are left in the [Failed to Notify](#) state after a failure.

1.3.3.3.5 Subordinate Transaction Manager Facet

The Subordinate Transaction Manager Facet provides the following services to [superior transaction manager facets](#):

- Acts as a Subordinate Transaction Manager Facet to a superior transaction manager facet in the Two-Phase Commit protocol.
- Provides recovery and outcome notification for transactions that are left in the [In Doubt](#) state after a failure.

1.3.4 Transaction Recovery

The atomicity property of a transaction guarantees that all participants in the transaction receive the same outcome. This guarantee is relaxed in the case of volatile resources such as voters but is strictly honored for durable resource managers and transaction managers.

To honor this guarantee, transaction managers and durable resource managers have to be capable of recovering from **transient failures** that can occur, such as loss of transport connectivity or a software crash. The process of recovery involves reestablishing connectivity with other transaction

participants and exchanging the protocol messages that are required to synchronize all parties on the actual outcome of the transaction.

After a transient failure, the transaction manager reestablishes connectivity with the following parties:

- The superior transaction manager for each transaction for which the transaction manager was [In Doubt](#) at the time of the failure.
- The subordinate transaction managers for which the transaction manager was in the [Failed to Notify](#) state at the time of the failure.

After a transient failure, the resource manager reestablishes connectivity with the following parties:

- The superior transaction manager for each transaction for which the resource manager was In Doubt at the time of the failure.

The following sections describe in more detail the recovery process for each participant.

1.3.4.1 Relationship Between Recovery and Durability

Transaction managers and durable resource managers can use any mechanism they choose to implement the durability guarantees of an atomic transaction.

At minimum:

- Before a durable resource manager or subordinate transaction manager sends a Prepared notification to its superior transaction manager, it MUST first ensure that it can derive the information that is needed to contact its superior transaction manager and to inquire about the outcome of the transaction after a transient failure. This requirement is needed for the subordinate to perform recovery on [In Doubt](#) transactions.
- Before a transaction manager can communicate the transaction outcome to a subordinate participant or the root application, it has to first ensure that it can derive the transaction outcome for as long as at least one durable subordinate has not acknowledged receipt of the transaction outcome. This requirement is needed for the superior to perform recovery on Failed to Commit transactions.
- Before a durable resource manager or subordinate transaction manager acknowledges a Commit notification from its superior transaction manager, it has to first ensure that it will not perform recovery on the transaction after a transient failure. This requirement allows the superior transaction manager to implement the **Presumed Abort** optimization.

The information that is needed in order to be able to contact another participant is identical to the information that was needed to establish the initial transport session with that participant as specified in [\[MS-CMPO\]](#) section 1.3.3.1.

1.3.4.2 Resource Manager Recovery

Resource manager recovery is unidirectional: the resource manager is always responsible for initiating recovery with its transaction manager. A resource manager always performs recovery on startup, even when it has not detected any transactions remaining in the [In Doubt](#) state. This is because the transaction manager cannot determine when it has [Failed to Notify](#) the resource manager of specific transaction outcomes.

The typical sequence for recovery of a resource manager is as follows:

1. The resource manager determines the list of transactions for which it is In Doubt. These are the transactions for which it previously voted Prepared but has not yet learned the outcome.
2. The resource manager registers with its transaction manager.
3. For each In Doubt transaction, the resource manager attempts to contact the transaction manager in order to determine the transaction outcome.
4. When the resource manager receives the outcome from the transaction manager, it performs any implementation-specific actions that are required to honor the ACID properties. Also, this process can take some time because the transaction manager can be acting as a subordinate transaction manager and it too might still be In Doubt about the actual transaction outcome.
5. After the resource manager ensures that there are no transactions for which it is still In Doubt, it informs the transaction manager that its recovery is complete. This allows the transaction manager to clean up any pending transactions for which it considered that the resource manager was in the Failed to Notify state.

1.3.4.3 Transaction Manager Recovery

Transaction manager recovery is dual-faceted. The recovering transaction manager will attempt to recover those transactions for which it is acting as a [superior transaction manager facet](#) and those for which it is acting as a [subordinate transaction manager facet](#).

The typical sequence for a superior transaction manager facet to perform recovery is the following:

1. The superior transaction manager determines the list of transactions for which it is in the [Failed to Notify](#) state. These are the transactions whose outcome has been decided but for which there exists at least one durable subordinate participant whose receipt of that outcome cannot be verified.
2. For each of these transactions, the superior transaction manager attempts to perform recovery by contacting all subordinate transaction managers whose receipt of outcome cannot be verified in order to redeliver the transaction outcome.

The typical sequence for a subordinate transaction manager facet to perform recovery is the following:

1. The subordinate transaction manager determines the list of transactions for which it is in the [In Doubt](#) state.
2. For each of these transactions, the subordinate transaction manager attempts contact the superior transaction manager in order to determine the transaction outcome.
3. For each In Doubt transaction whose transaction outcome is now known, the subordinate transaction manager proceeds to communicate the outcome to its own subordinate transaction managers.

1.3.5 Transaction Propagation

A single transaction typically requires work to be performed by one or more resource managers for one or more applications. Each of these applications and resource managers is typically associated with exactly one transaction manager.

When two participants share a common transaction manager, all that is needed to share a transaction is agreement on the transaction's unique identifier. How this unique identifier is communicated among the applications and resource managers is implementation-specific.

However, when two participants do not share a common transaction manager, this protocol defines a propagation mechanism that enables the two participants to notify their respective transaction managers that a specified transaction will span the two transaction managers. Transaction propagation allows applications and resource managers to freely marshal transactions across process and host machine boundaries by using whatever communication mechanisms and formats they chose.

When a participant (the source) determines that it marshals a transaction to a second participant (the destination), the participant chooses between two distinct propagation techniques:

- Push propagation
- Pull propagation

Push propagation requires the source participant to have a prior knowledge about which transaction manager the destination participant is associated with (as specified in section [1.3.5.2](#)). In contrast, pull propagation allows the source participant to marshal the transaction without any awareness of the transaction manager of the destination participant (as specified in section [1.3.5.1](#)).

Independent of the choice of push or pull propagation, after the propagation is complete, the destination transaction manager will have enlisted with the source transaction manager to coordinate the outcome of the transaction. In this enlistment, the source transaction manager plays the role of superior transaction manager, and the destination transaction manager plays the role of subordinate transaction manager.

1.3.5.1 Pull Propagation

Pull propagation enables the untargeted marshaling of a transaction from one application or resource manager to another. Contact information for the destination transaction manager is not required to be known by the source in advance.

The following sequence of events represents a complete pull propagation operation between two participants:

- When the source determines that it possesses a transaction that it wants to share with the destination, it provides the destination with marshaling information about the transaction being shared in an implementation-specific manner. The marshaling information needs to be sufficient for the destination to create a [Propagation Token](#) structure, as specified in section [2.2.5.4](#), that corresponds to the transaction being shared.
- The destination contacts its own transaction manager and requests that it join the transaction by using the marshaling information that is provided by the source application.
- If the destination transaction manager is not already a participant in the transaction, the destination transaction manager uses the marshaling information to contact the source transaction manager to enlist in the transaction as a subordinate transaction manager. This inter-transaction manager handshake is called pull propagation.
- If the operation is successful, the destination transaction manager reports success to the destination. The destination performs further operations on the transaction with its associated transaction manager or marshals the transaction further to other participants.

The following figure shows a typical pull propagation.

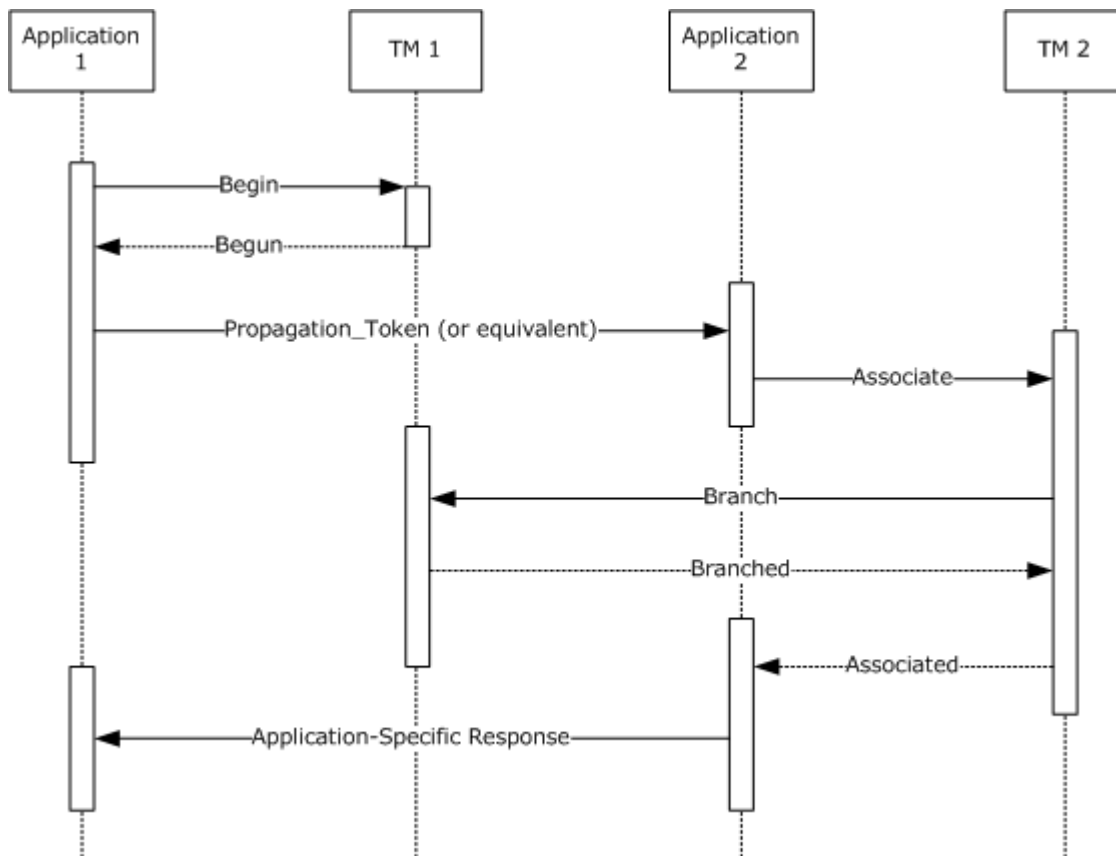


Figure 9: Transaction manager pull propagation

1.3.5.2 Push Propagation

Push propagation enables the targeted marshaling of a transaction from one participant to another. Push propagation is available only when the source knows the contact information for the destination transaction manager in advance.

Push propagation consists of two distinct logical operations: an export operation and an import operation.

The following sequence of events represents a complete push propagation operation between two participants:

- The source obtains contact information for the destination transaction manager by using implementation-specific means. The contact information consists of whatever the source needs to construct an [SWhereabouts](#) structure, as specified in section [2.2.5.11](#). This step need only be performed one time per destination, because the contact information is not specific to a specified transaction or propagation.
- When the source determines that it possesses a transaction that it wants to share with the destination, the source asks its transaction manager to export the transaction to the destination transaction manager by using the contact information it obtained in the previous step.

- The source transaction manager contacts the destination transaction manager by using the provided contact information and informs it of the existence and details of the transaction. This inter-transaction manager handshake is the export operation of push propagation.
- When the source transaction manager receives acknowledgment from the destination transaction manager, the export operation is complete. If the destination transaction manager was not already a participant in the transaction, the destination transaction manager is now enlisted as a subordinate transaction manager at the source transaction manager, which acts as the superior transaction manager.
- After the source transaction manager informs the source that the transaction was successfully exported, the source then uses an implementation-specific mechanism to marshal the exported transaction to the destination. The marshaled information MAY take any form that the source and destination agree on, but MUST be sufficient for the source to construct an [STxInfo](#) structure as specified in section [2.2.5.10](#).
- The destination uses the marshaled information that is provided by the source to request an import operation from its transaction manager. The import operation is typically a simple confirmation that the transaction exists and was correctly exported to the destination.
- If the import operation is successful, the destination transaction manager reports success to the destination. The destination performs further operations on the transaction with its associated transaction manager or marshals the transaction further to other participants.

The following figure depicts a typical push propagation.

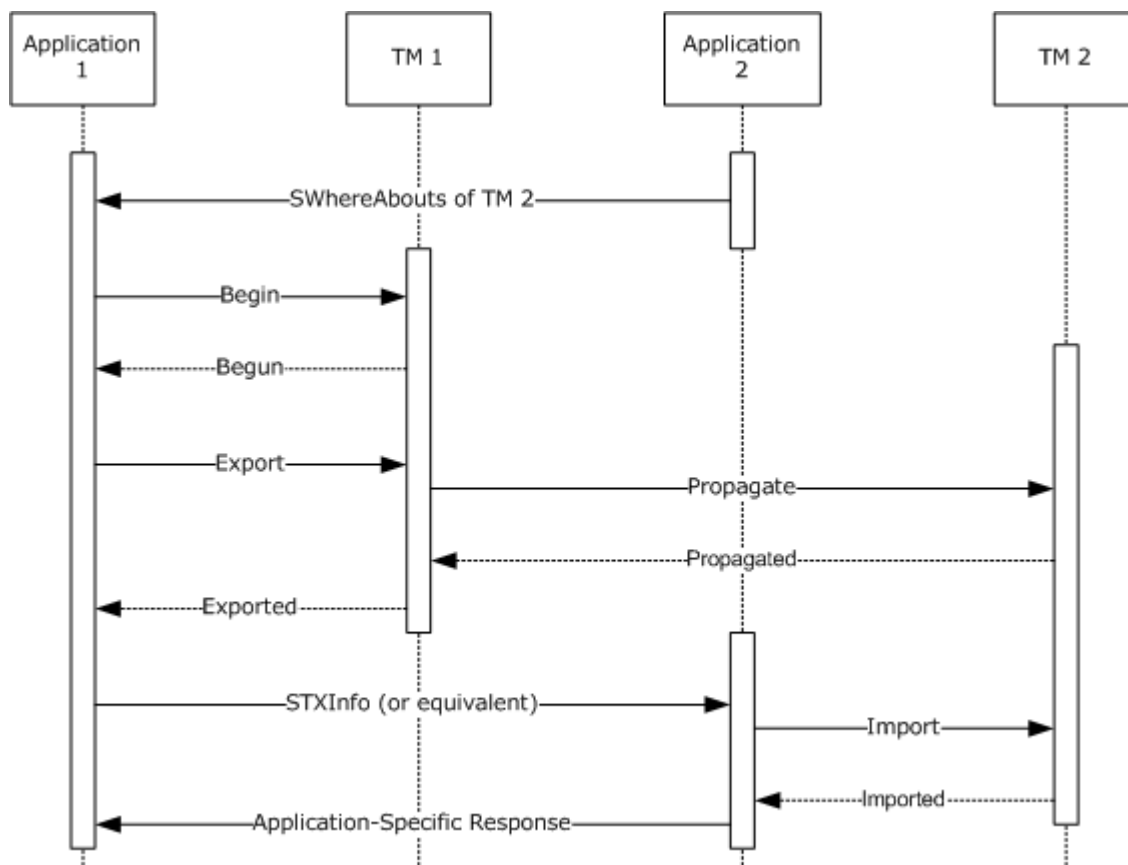


Figure 10: Transaction manager push propagation

1.4 Relationship to Other Protocols

This protocol relies on the session and **connection** transport infrastructure as specified in [\[MS-CMPO\]](#) section 3.2.1.2 and [\[MS-CMP\]](#) section 3.1.1.1. The following diagram illustrates the protocol layering for this protocol:

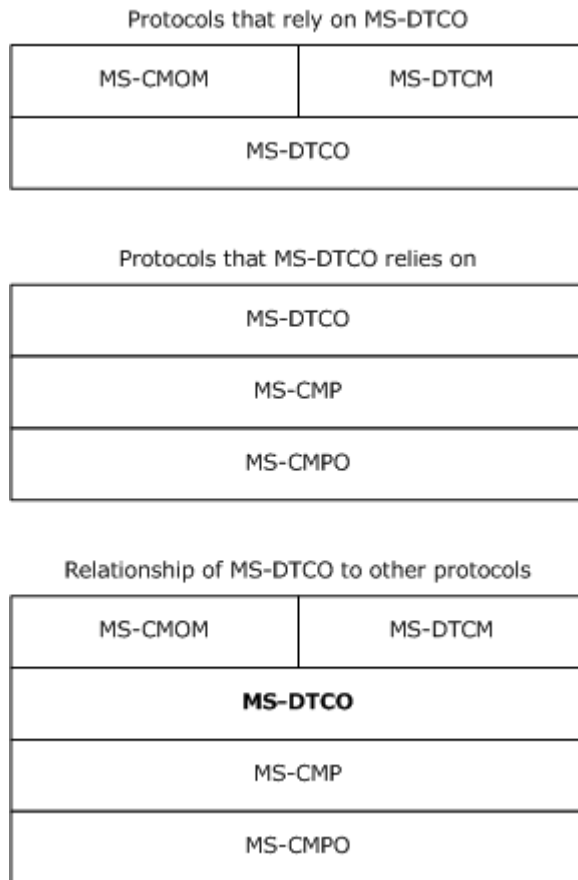


Figure 11: Protocol relationships

This protocol provides extensibility elements that are used by the following specifications:

- [\[MS-CMOM\]](#)
- [\[MS-DTCM\]](#)

The following protocols perform transaction marshaling by using the structures that are specified in section [2.2.5](#) of this document:

- [\[MS-COM\]](#)
- [\[MS-MQRR\]](#)
- [\[MS-MQMP\]](#)

1.5 Prerequisites/Preconditions

This protocol requires that all participating roles possess implementations of the [MSDTC Connection Manager: OleTx Transports Protocol Specification](#) (specified in [MS-CMPO]) and [MSDTC Connection Manager: OleTx Multiplexing Protocol Specification](#) (specified in [MS-CMP]) protocols.

1.6 Applicability Statement

This protocol applies to scenarios where distributed atomic transaction processing is required.

Distributed transactions are generally required in scenarios where a number of applications and resource managers cooperate to perform a set of related work items that require the ACID properties of a distributed transaction. These properties are needed in order to make changes to persistent state in a deterministic, correct, and highly reliable manner. Although distributed transactions are one of several mechanisms for accomplishing this goal, they are the most efficient and understood general-purpose solution.

This particular distributed transaction protocol requires network topologies where the [\[MS-CMPO\]](#) and [\[MS-CMP\]](#) protocols constitute a viable network transport for establishing long-lived session relationships between different parties supporting many short-lived connection exchanges that accomplish specific tasks.

1.7 Versioning and Capability Negotiation

This document covers versioning aspects in the following areas:

- Protocol versions

This protocol provides four different versions: 1, 2, 4, and 5 (version 3 is reserved and not used). More details on the protocol elements supported in each version are provided in [Protocol Versioning \(section 2.2.1\)](#).

- Capability negotiation

This protocol performs explicit versioning and capability negotiation, as specified in sections [1.7.2](#) and section [1.7.3](#).

1.7.1 Versioning Mechanisms

This protocol uses various mechanisms for versioning that are introduced below.

- Protocol Version Numbers as versioning mechanism:

This protocol provides four different versions. The following are the implications of supporting a particular protocol version:

- Support for connection types is version specific and is either required, optional, or not allowed for a given Protocol Version.
- Support for messages is version specific and is either required, optional or not allowed for a given Protocol Version.
- The layout of data associated with specific messages is version specific and is determined by the Protocol Version.

[Protocol Version Numbers as a Versioning Mechanism \(section 2.2.1.1\)](#) specifies details of what it means to support a certain Protocol Version Number. [Protocol Versioning Details \(section 3.1.4\)](#) specifies how the Protocol Version numbers are negotiated during communication initiation.

- Structures with fields containing version numbers as versioning mechanism: Certain structures have fields containing version numbers that specify how to interpret other parts of the structure. As an example the [Propagation Token \(section 2.2.5.4\)](#) structure has the fields **dwVersionMin** and **dwVersionMax** the values of which are used to indicate whether certain other fields are present or not.

[Structures with Fields Containing Version Numbers as Versioning Mechanism \(section 2.2.2\)](#) provides a list of the structures that fall in this category and links to information regarding each.

- Structures with complex fields using specific values to indicate the type of the complex field. Certain structures have a field that specifies how to interpret other parts of the structure. As an example, the [STmToTmProtocol structure \(section 2.2.5.9\)](#) uses the value of the **tmprotDescribed** field to specify how to interpret the rest of the fields in that structure.

[Structures with a Format-Specifying Field as Versioning Mechanism \(section 2.2.3\)](#) provides a list of the structures that fall in this category and links to information regarding each.

1.7.2 Versioning Negotiation Mechanisms

This protocol uses the following versioning negotiation mechanisms for each of the versioning mechanisms discussed above.

- Protocol Version Numbers as versioning mechanism

This protocol makes use of the explicit versioning negotiation mechanism as specified in [\[MS-CMPO\]](#), [BuildContext Primary](#), section 3.1.4.2.1. An implementation of this protocol uses this mechanism to specify which versions of the protocol it supports and to negotiate a mutually agreeable version with its partners (see [Protocol Versioning Details, \(section 3.1.4\)](#)).

- Structures with fields containing version numbers as versioning mechanism

There is no versioning negotiation mechanism for this case. The version numbers are passed in each structure by the sender, and interpreted by the receiver.

- Structures with a field containing a value that identifies the structure format as versioning mechanism

There is no versioning negotiation mechanism for this case. The values of the field specifying the format are passed in each structure by the sender, and interpreted by the receiver.

1.7.3 Capability Negotiation Mechanisms

This protocol uses the following capability negotiation mechanisms for each of the versioning mechanisms discussed above.

- Protocol Version Numbers as versioning mechanism:
 - Support for certain connection types is optional for a specific protocol version. A connection **initiator** can determine whether the **acceptor** supports these connection types by sending the first message for the connection and determining the acceptor's level of support from the response. If the acceptor rejects the connection with a [MTAG_CONNECTION_REQ_DENIED](#) as specified in ([\[MS-CMP\]](#) section 2.2.5), the connection type is not supported.

- Support for a message type is never optional for a specific protocol version, with one exception: [TXUSER_RESOLVE_MTAG_ACCESSDENIED \(section 2.2.8.3.2.1\)](#). However there is no negotiation process to determine support for this message, and the message is sent by a sender that supports it in all cases.
- Some specific data fields inside certain message types were added in specific protocol versions as additional data fields that appear after the fields that are defined by previous protocol versions. The receivers examine the size of the incoming [MESSAGE_PACKET \(section 2.2.4.1\)](#) structure to determine which additional data fields, if any, were included in the message by the sender.

- Structures with fields containing version numbers as versioning mechanism.

The structures using version numbers as versioning mechanism do not have any optional elements for a particular version. Therefore there are no capability negotiation mechanisms associated with them.

- Structures with a field containing a value that identifies the structure format as a versioning mechanism.

In this case, the format of the structure is completely determined by the respective format-specifying field. There are no capability negotiation mechanisms associated with these structures.

1.8 Vendor-Extensible Fields

MSDTC Connection Manager: OleTx Transaction Protocol gives vendors the ability to provide implementation-specific **protocol extensions** to the [Core Transaction Manager Facet](#). This protocol provides the following vendor-extensible fields and data elements:

- A protocol extension provides a set of services, as specified in section [3.2.1.6](#).
- A protocol extension also includes the contribution of **extended whereabouts** information to the Core Transaction Manager Facet, as specified in section [3.2.3](#).
- A protocol extension can augment the default set of transaction manager facets that are implemented inside an implementation of the transaction manager role, as specified in sections [3.2.1.5](#) and [3.2.3](#). Each vendor-supplied transaction manager facet processes the events as specified in section [3.2.4](#).
- Each vendor-supplied transaction manager facet has the option to use the local events that are provided by the Core Transaction Manager Facet that is specified in [3.2.7](#).

1.9 Standards Assignments

This protocol has no standards assignments.

2 Messages

The following sections specify how MSDTC Connection Manager: OleTx Transaction Protocol messages are transported. These sections also define how this protocol maps over lower-layer protocols, and they also define the syntax for the messages that are used by this protocol.

2.1 Transport

This protocol uses implementations of the [MSDTC Connection Manager: OleTx Transports Protocol Specification](#) (specified in [MS-CMPO]) and [MSDTC Connection Manager: OleTx Multiplexing Protocol Specification](#) (specified in [MS-CMP]) protocols as the transport layer for sending and receiving protocol messages.

2.1.1 Messages, Connections, and Sessions

The layout of each message that is defined by this protocol MUST extend the [MESSAGE PACKET](#) structure, as specified in section [2.2.4.1](#). The general mechanisms that are used to send and receive messages are as specified in [\[MS-CMP\]](#) section 2.1.1.

Each message MUST be sent by using an active [MS-CMP] connection that has been established between an initiator and an acceptor. The mechanisms that are used to initiate and accept connections are as specified in [\[MS-CMP\]](#) section 3.1.4.

Each connection MUST be initiated inside an active [\[MS-CMPO\]](#) session that has been established between two OleTx participants. The mechanisms that are used to establish sessions are as specified in [\[MS-CMPO\]](#) section 1.3.3.

2.1.2 MS-CMPO Parameterization

In order to establish an [\[MS-CMPO\]](#) session as specified in [MS-CMPO], [Local Partner State](#) (section 3.2.1.1), the following values MUST be provided to the lower-layer protocol:

- A security-level value that indicates the RPC authentication level that is wanted. The possible values for this element are as specified in [MS-CMPO], Local Partner State, (section 3.2.1.1).
- The minimum and maximum protocol version values, as specified in [\[MS-CMPO\]](#) section **2.2.4**.
- A Name object that indicates the host name, the **contact identifier**, and the supported RPC network protocols of the remote **endpoint** against which the session is to be established. Name objects are as specified in [\[MS-CMPO\]](#) section 3.2.1.4.

2.1.2.1 Computing a Security Level

Every OleTx participant SHOULD use mutual authentication when establishing a new session. If the destination does not support mutual authentication, a participant SHOULD use incoming authentication. If the destination does not support incoming authentication, a participant MAY use no security. [<1>](#)

2.1.2.2 Computing Protocol Version Values

The process for computing the minimum and maximum protocol version numbers used in initializing the underlying [\[MS-CMPO\]](#) transport is defined in [Protocol Versioning Details \(section 3.1.4\)](#).

2.1.2.3 Computing a Name Object

The Name object that is used to initiate a session is obtained in a variety of ways. This section defines how to obtain the appropriate Name object for several common situations. The specific transaction processing roles mentioned in these sections (applications, resource managers, and transaction managers) are defined as specified in section [3.2.3](#).

When an application or resource manager initiates a connection to its transaction manager, the application or resource manager **MUST** use implementation-specific configuration information to compute a Name object that represents the transaction manager:

1. For pull propagation of transactions, the source application **MUST** include the Name object representing its transaction manager in the marshaling information that is sent to the destination application. The [Propagation Token](#) structure, as specified in section [2.2.5.4](#), **SHOULD** be used as the transfer syntax for this marshaling information.
2. For pull propagation of transactions, the subordinate transaction manager (the transaction manager of the destination) **MUST** communicate its own Name object to the superior transaction manager (the source transaction manager) using a `CONNTYPE_PARTNERTM_BRANCH` connection.
3. For push propagation of transactions, the destination application **MUST** make the Name object that represents its transaction manager available to the source application. The [SWhereabouts](#) structure that is defined as specified in section [2.2.5.11](#) **SHOULD** be used as the transfer syntax for this information. Alternatively, the [NAMEOBJECTBLOB](#) structure that is defined as specified in section [2.2.5.3](#), **MAY** be used for the same purpose. <2>
4. For push propagation of transactions, the superior transaction manager **MUST** communicate its own Name object to the subordinate transaction manager using a [CONNTYPE_PARTNERTM_PROPAGATE](#) (section [2.2.9.1.1.1](#)) connection.

2.2 Message Syntax

2.2.1 Protocol Versioning

2.2.1.1 Protocol Version Numbers as a Versioning Mechanism

This protocol has four versions: 1, 2, 4, and 5 (version 3 is reserved and not used). For each version, there is a set of protocol elements that **MUST** be supported (called version-required elements), a set of optional protocol elements that **SHOULD** be supported (called version-optional elements), and a set of protocol elements that **MUST NOT** be supported. The following sections provide versioning tables that specify the scope of each protocol version with respect to the three mentioned sets.

The tables contain the following values.

Value	Description
Yes	The protocol element MUST be supported in the respective protocol version.
No	The protocol element MUST NOT be supported in the respective protocol version.
Optional	The protocol element SHOULD be supported in the respective protocol version.

2.2.1.1.1 Version-Specific Aspects of Connection Types Relevant to an Application

The following table shows version-specific aspects for connection types that are relevant to applications. This table includes connection types and messages that are supported on certain versions as well as messages whose size is version specific. If a connection type or message that is relevant to applications is omitted from this table, it is not version specific and **MUST** be supported on all versions.

Version-specific aspect	V1	V2	V4	V5
Version supports connection type CONNTYPE_TXUSER_BEGIN2 (section 2.2.8.1.2) .	No	Yes	Yes	Yes
Version supports connection type CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS (section 2.2.8.2.2.1) .	No	No	No	Optional <3>
Version supports connection type CONNTYPE_TXUSER_GETSECURITYFLAGS (section 2.2.8.4.1) .	No	No	Yes	Yes
Version supports connection type CONNTYPE_TXUSER_IMPORT2 (section 2.2.8.2.2.4) .	No	Yes	Yes	Yes
Version supports connection type CONNTYPE_TXUSER_PROMOTE (section 2.2.8.1.3) .	No	No	No	Optional <4>
Version supports connection type CONNTYPE_TXUSER_SETTXTIMEOUT (section 2.2.8.3.3) .	No	Optional <5>	No	No
Version supports connection type CONNTYPE_TXUSER_SETTXTIMEOUT2 (section 2.2.8.3.4) .	No	No	Optional	Optional <6>
Version supports connection type CONNTYPE_TXUSER_TRACE (section 2.2.8.3.5) .	No	No	Yes	Yes
Version supports messages TXUSER_EXPORT_MTAG_CREATE2 (section 2.2.8.2.2.2.2) and TXUSER_EXPORT_MTAG_CREATE_NET_TX_DISABLED (section 2.2.8.2.2.2.4) .	No	No	Yes	Yes
Version supports message TXUSER_RESOLVE_MTAG_ACCESSDENIED (section 2.2.8.3.2.1) .	No	No	Optional <7>	Yes
The SourceTmAddress field is described by the structure NAMEOBJECTBLOB (section 2.2.5.3) in message TXUSER_ASSOCIATE_MTAG_ASSOCIATE (section 2.2.8.2.1.1.1) .	Yes	No	No	No
The SourceTmAddress field is described by the structure OLETX_TM_ADDR (section 2.2.4.2) in message TXUSER_ASSOCIATE_MTAG_ASSOCIATE (section 2.2.8.2.1.1.1).	No	Yes	Yes	Yes
The SourceTmAddress field is described by the structure NAMEOBJECTBLOB (section 2.2.5.3) in message TXUSER_EXPORT_MTAG_CREATE (section 2.2.8.2.2.2.1) .	Yes	No	No	No

Version-specific aspect	V1	V2	V4	V5
The SourceTmAddress field is described by the structure OLETX_TM_ADDR (section 2.2.4.2) in message TXUSER_EXPORT_MTAG_CREATE (section 2.2.8.2.2.2.1).	No	Yes	Yes	Yes
The grfNetworkDtcAccess field of the TXUSER_GETSECURITYFLAGS_MTAG_FETCHED message supports (uses) the following DTCADVCONFIG bits: DTCADVCONFIG_NETWORKDTCACCESS_ENABLE DTCADVCONFIG_NETWORKDTCACCESS_ADMIN DTCADVCONFIG_NETWORKDTCACCESS_TX DTCADVCONFIG_NETWORKDTCACCESS_CLIENTS DTCADVCONFIG_NETWORKDTCACCESS_TIP	No	No	Yes	Yes
The grfNetworkDtcAccess field of the TXUSER_GETSECURITYFLAGS_MTAG_FETCHED message supports (uses) the following DTCADVCONFIG bits: DTCADVCONFIG_INBOUNDNETWORK_TX DTCADVCONFIG_OUTBOUNDNETWORK_TX DTCADVCONFIG_SECURITYLEVEL_NOSECURITY DTCADVCONFIG_SECURITYLEVEL_AUTHENTICATEDONLY DTCADVCONFIG_SECURITYLEVEL_MUTUALAUTH	No	No	No	Yes
The guidSignature field in the STxInfo structure (present in propagation-related messages) uses a reserved GUID	No	Yes	Yes	Yes
The STxInfo structure supports versioning based on its guidSignature field	No	Yes	Yes	Yes

2.2.1.1.2 Version-Specific Aspects of Connection Types Relevant to a Transaction Manager

The following table shows version-specific aspects for connection types that are relevant to transaction managers. This table includes connection types and messages that are supported on certain versions as well as messages whose size is version specific. If a connection type or message that is relevant to transaction managers is omitted from this table, it is not version specific and MUST be supported on all versions.

Version-specific aspect	V1	V2	V4	V5
PARTNERTM_PROPAGATE_MTAG_PHASE0 , PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE , PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER , PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTERED , and PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTRATIONREJECTED .	N	Y	Y	Y

2.2.1.1.3 Version-Specific Aspects of Connection Types Relevant to a Resource Manager

The following table shows version-specific aspects for connection types that are relevant to resource managers. These include connection types and messages that are supported on certain [MS-DTCO]

versions as well as messages whose size is version specific. If a connection type or message that is relevant to resource managers is omitted from this table, then it is not version specific and **MUST** be supported on all versions.

Version-specific aspect	V1	V2	V4	V5
Version supports connection type CONNTYPE_TXUSER_PHASE0 .	No	Yes	Yes	Yes
Version supports connection type CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL .	No	No	No	Optional<8>

2.2.2 Structures with Fields Containing Version Numbers as Versioning Mechanism

Currently, only one structure has fields that specify the version (and therefore the format) of the structure.

Structure	Fields containing version numbers
Propagation Token	dwVersionMin dwVersion Max

2.2.3 Structures with a Format-Specifying Field as Versioning Mechanism

The following table contains the structures that have a field whose value indicates the format of the structure.

Structure	Format-specifying field
STmToTmProtocol	tmprotDescribed
STxInfo	guidSignature

2.2.4 Common Structures

2.2.4.1 MESSAGE_PACKET

The MESSAGE_PACKET structure defines the initial message fields that are contained by all **MTAGs** in this protocol, as specified in [\[MS-CMP\]](#) section 2.2.2.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgTag																															
fIsMaster																															
dwConnectionId																															
dwUserMsgType																															
dwcbVarLenData																															
dwReserved1																															

MsgTag (4 bytes): A 4-byte integer value that describes the OLE transaction message type. For all uses in this document, this value MUST be MTAG_USER_MESSAGE, as specified in [\[MS-CMP\]](#) section 2.2.8.

fIsMaster (4 bytes): A 4-byte value indicating the direction of the message in the conversation.

This value MUST be one of the following values.

Value	Meaning
0x00000000	The message is sent by the party that accepted the connection.
0x00000001	The message is sent by the party that initiated the connection.

dwConnectionId (4 bytes): A 4-byte integer value that MUST contain the unique identifier for the associated connection.

dwUserMsgType (4 bytes): This field contains the message type identifier. Each MTAG that is defined in this section MUST specify a distinct value for this field for a specified connection type.

dwcbVarLenData (4 bytes): An unsigned 4-byte integer value that MUST contain the size, in bytes, of the message buffer that contains the MESSAGE_HEADER structure, minus the size, in bytes, of the MESSAGE_HEADER structure itself.

dwReserved1 (4 bytes): Reserved. This value MUST be set to an implementation-specific value and MUST be ignored on receipt. [<9>](#)

2.2.4.2 OLETX_TM_ADDR

The OLETX_TM_ADDR structure is used to represent the address of a transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
guidSignature																															
...																															
...																															
...																															
guidEndpoint																															
...																															
...																															
...																															
grbComProtsSupported																															
wszHostName (variable)																															
...																															

guidSignature (16 bytes): This field contains a signature value for this structure. The value MUST be the binary representation of the **GUID** {DC85CB48-D8A5-11d2-828B-00805F0DF75A}.

guidEndpoint (16 bytes): This field MUST contain a GUID that specifies the contact identifier of the transaction manager.

grbComProtsSupported (4 bytes): Indicates the RPC transports for which the transaction manager is listening. The value MUST be a bitwise OR operator of one or more [\[MS-CMPO\]](#) flags, as specified in [\[MS-CMPO\]](#), section [2.2.7](#).

wszHostName (variable): This field MUST contain a null-terminated, little-endian UTF-16 encoded string that specifies the NetBIOS host name of the transaction manager. This field MUST NOT contain a **Unicode** byte-order-mark (BOM) character. The length of this field MUST be 2 to 32 bytes, inclusive.

2.2.4.3 OLETX_VARLEN_STRING

The OLETX_VARLEN_STRING structure is used to represent a byte-counted variable length string.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
cbLength																															
szString (variable)																															
...																															

cbLength (4 bytes): An unsigned integer that MUST contain the number of bytes in the **szString** field.

szString (variable): A Latin-1 string as specified in [ISO-8859-1](#) without a final null-terminating character. This field MUST be **cbLength** bytes in length. If **cbLength** is zero, this field MUST NOT be present.

2.2.5 Transaction Propagation Structures

2.2.5.1 Associate_Msg_Version2

The Associate_Msg_Version2 structure contains the NetBIOS host name of a transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
cbHostNameW																															
wszHostName (variable)																															
...																															

cbHostNameW (4 bytes): The size, in bytes, of **wszHostName**, including the null terminator. The value of this field MUST be in the range 2 to 32 bytes, inclusive.

wszHostName (variable): A null-terminated, little-endian UTF-16 encoded string that contains a NetBIOS host name. This string MUST have the length that is specified by **cbHostNameW** and MUST NOT contain a Unicode byte-order-mark (BOM) character.

2.2.5.2 Associate_Msg_Version3

The Associate_Msg_Version3 structure contains information about the transaction protocol support of a transaction manager. [<10>](#)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
fNetworkTxEnabled																															
fTipEnabled																															
cbTipTmUrl																															
szTipTmUrl (variable)																															
...																															

fNetworkTxEnabled (4 bytes): This field indicates if network access is enabled or disabled on the transaction manager. If network access is disabled, this field MUST be set to zero. If network access is enabled, this field MUST be set to a nonzero value.

fTipEnabled (4 bytes): This field indicates if the transaction Internet Protocol (TIP) is enabled or disabled on the transaction manager, as specified in [RFC2371](#). If TIP is disabled, this field MUST be set to zero. If TIP is enabled, this field MUST be set to a nonzero value. For more information about the TIP protocol, see [RFC2371](#) for details.

cbTipTmUrl (4 bytes): This field MUST contain the size, in bytes, of **szTipTmUrl**, including the null terminator. The value of this field MUST be greater than or equal to 0.

szTipTmUrl (variable): A null-terminated Latin-1 ANSI string, as specified in [ISO-8859-1](#), that MUST contain the URL of the TIP transaction manager on the node that created this propagation token. If cbTipTmUrl is zero, this field MUST NOT be present. Otherwise, this field MUST have the length specified by **cbTipTmUrl**.

2.2.5.3 NAMEOBJECTBLOB

The NAMEOBJECTBLOB structure contains information to identify and locate a transaction manager.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
szGuid																															
...																															
...																															
...																															
...																															
...																															
...																															
(szGuid cont'd for 2 rows)																															
dwcbHostName																															
dwReserved1																															
grbComProtsSupported																															
szHostName (variable)																															
...																															

szGuid (40 bytes): A fixed-size array, containing a null-terminated Latin-1 ANSI string as specified in [\[ISO-8859-1\]](#), that contains a GUID that is formatted into a string, as specified in [\[C706\]](#), [Appendix A, UUID](#). This string MUST identify the contact identifier for the transaction manager instance that is located at the node that is identified by the host name. Storage after the initial null MUST be ignored on receipt.

dwcbHostName (4 bytes): This field MUST contain the size, in bytes, of the **szHostName** field, including the null terminator. The value of this field MUST be in the range 1 to 16, inclusive.

dwReserved1 (4 bytes): Reserved. This field MUST be set to an implementation-specific value, and MUST be ignored on receipt. [<11>](#)

grbComProtsSupported (4 bytes): Indicates which RPC transports the transaction manager is able to use to communicate. The value MUST be a bitwise OR operator of one or more flags, as specified in [\[MS-CMPO\]](#). The [COM_PROTOCOL](#) data type is implemented as specified in [\[MS-CMPO\]](#) section 2.2.7.

szHostName (variable): A null-terminated Latin-1 ANSI string, as specified in [\[ISO-8859-1\]](#), that MUST specify the host name of the transaction manager instance. It MUST have the length specified by **dwcbHostName**.

2.2.5.4 Propagation_Token

The Propagation Token structure is used for performing pull-based transaction propagation. This structure contains information about a transaction and about a superior transaction manager that is available for use by participants to enlist on the transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
dwVersionMin																															
dwVersionMax																															
guidTx																															
...																															
...																															
...																															
isoLevel																															
isoFlags																															
cbSourceTmAddr																															
szDesc																															
...																															
...																															
...																															
...																															

...
...
...
(szDesc cont'd for 2 rows)
NameObject (variable)
...
AssociateMsgVersion2 (variable)
...
AssociateMsgVersion3 (variable)
...

dwVersionMin (4 bytes): The minimum version of the transaction information structure that accompanies the propagation token. The value MUST be set to 1.

dwVersionMax (4 bytes): The maximum version of the transaction information structure that accompanies the propagation token. The value MUST be 1, 2, or 3. [<12>](#12)

guidTx (16 bytes): This field MUST contain a GUID that specifies the **transaction identifier**.

isoLevel (4 bytes): The isolation level of the transaction. This field MUST contain one value from the [OLETX ISOLATION LEVEL](#) enumeration.

isoFlags (4 bytes): The isolation flags for the transaction. This field MUST contain the bitwise OR operator of zero or more values from the [OLETX ISOLATION FLAGS](#) enumeration.

cbSourceTmAddr (4 bytes): This field MUST contain the total size, in bytes, of the space that is used by the **NameObject**, **AssociateMsgVersion2**, and **AssociateMsgVersion3** fields.

szDesc (40 bytes): The description of the transaction, as a fixed-size array of 40 bytes containing a null-terminated Latin-1 ANSI string, as specified in [\[ISO-8859-1\]](#). This field MUST be set to an implementation-specific value. Any bytes that follow the first null-terminator character SHOULD be set to zero, and MUST [<13>](#13) be ignored on receipt.

NameObject (variable): This field MUST be a [NAMEOBJECTBLOB](#) structure that contains contact information about the transaction manager that is referenced by the propagation token. If **dwVersionMax** is 1, then this field MUST NOT be present; otherwise, it MUST be present.

AssociateMsgVersion2 (variable): This field MUST be an [Associate Msg Version2](#) structure that contains the NetBIOS host name for the transaction instance that is referenced by the

Propagation Token. If **dwVersionMax** is 1, then this field MUST NOT be present; otherwise, it MUST be present. If this field is present, the contents MUST override the **szHostName** value in the **NameObject** field.

AssociateMsgVersion3 (variable): This field MUST be an [Associate Msg Version3](#) structure that contains information about the transaction protocol support for the transaction manager that is referenced by the Propagation Token. If **dwVersionMax** is 3, then this field MUST be present; otherwise, it MUST NOT be present.

2.2.5.5 SDtcCmEndpointInfoV1

The SDtcCmEndpointInfoV1 structure contains data used to connect to a transaction manager that supports the OleTx protocol.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
comprotSupported																															
guidEndpointID																															
...																															
...																															
...																															
szHostname (variable)																															
...																															

comprotSupported (4 bytes): Indicates which RPC transports the transaction manager supports for communication. The value MUST be a bitwise OR operator of one or more [\[MS-CMPO\]](#) flags. The [COM_PROTOCOL](#) data type is implemented as specified in [\[MS-CMPO\]](#) section 2.2.7.

guidEndpointID (16 bytes): This field MUST be a GUID that specifies the contact identifier of the transaction manager.

szHostname (variable): A null-terminated Latin-1 ANSI character string, as specified in [\[ISO-8859-1\]](#), that MUST specify the host name for the transaction manager endpoint. This field MUST be between 1 and 16 bytes in length, inclusive.

2.2.5.6 SDtcCmEndpointInfoV2

The SDtcCmEndpointInfoV2 structure contains extended information that is used, along with the contents of the [SDtcCmEndpointInfoV1 \(section 2.2.5.5\)](#) structure, to connect to a transaction manager that supports the OleTx protocol.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
wszHostname (variable)																															
...																															

wszHostname (variable): A null-terminated little-endian UTF-16 character string that specifies the NetBIOS host name for the transaction manager endpoint. This field **MUST** be between 2 and 32 bytes in length, inclusive, and **MUST NOT** contain a Unicode byte-order-mark (BOM) character.

2.2.5.7 SOleTxInfoForTip

The SOleTxInfoForTip structure contains data that is specific to the Transaction Internet Protocol (TIP) for an exported transaction.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
szDescription																															
...																															
...																															
...																															
...																															
...																															
...																															
(szDescription cont'd for 2 rows)																															
isoLevel																															
isoFlags																															
szTipTmUrl (variable)																															
...																															

szDescription (40 bytes): See the **szDesc** field in [Propagation Token \(section 2.2.5.4\)](#) for details.

isoLevel (4 bytes): The isolation level of the transaction. The value MUST be one as specified in the [OLETX ISOLATION LEVEL \(section 2.2.6.9\)](#) enumeration.

isoFlags (4 bytes): The isolation flags for the transaction. The value MUST be a legal combination of values from the [OLETX ISOLATION FLAGS \(section 2.2.6.8\)](#) enumeration.

szTipTmUrl (variable): A null-terminated Latin-1 ANSI string, as specified in [\[ISO-8859-1\]](#), that MUST specify the TIP URL of the transaction manager, as specified in [\[RFC2371\]](#).

2.2.5.8 SExtendedEndpointInfo

The SExtendedEndpointInfo packet contains data to represent endpoint information that is available for use to connect to a protocol extension that is hosted by a transaction manager. This structure

does not specify its own length. Therefore, it MUST be used in a context that specifies the actual length.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
guidProtocolExtension																															
...																															
...																															
rgbProtocolExtensionData (variable)																															
...																															

guidProtocolExtension (16 bytes): This field MUST contain a GUID that specifies the protocol extension that contributed this extended endpoint information.

rgbProtocolExtensionData (variable): This field MUST contain data that is contributed by a protocol extension that represents protocol extension-specific endpoint information. The format and size of this data is specific to the respective extension protocol. This data MUST NOT be interpreted by an application or other transaction participant unless it recognizes the **guidProtocolExtension** field.

2.2.5.9 STmToTmProtocol

The STmToTmProtocol structure contains data that is used by a particular type of transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
tmprotDescribed																															
cbTmProtocolData																															
rgbTmProtocolData (variable)																															
...																															

tmprotDescribed (4 bytes): Transaction-manager-to-transaction-manager protocol-specific data for this transaction. This MUST be one of the values specified in [TM Protocol \(section 2.2.6.2\).](#)<14>

cbTmProtocolData (4 bytes): This field MUST specify the length, in bytes, of the **rgbTmProtocolData** field.

rgbTmProtocolData (variable): The transaction manager protocol-specific data for this transaction. If the **cbTmProtocolData** field is 0, this field MUST NOT be present. Otherwise, the format of this field depends on the value of the **tmprotDescribed** field, which MUST be one of the following values.

tmprotDescribed name/value	Meaning
TmProtocolMsdtcV1 0x00000002	This field MUST contain an SDtcCmEndpointInfoV1 (section 2.2.5.5) structure that contains data that is used to connect to an OleTx transaction manager. The cbTmProtocolData field MUST be at least 21.
TmProtocolMsdtcV2 0x00000003	This field MUST contain an SDtcCmEndpointInfoV2 (section 2.2.5.6) structure that contains additional data that is used to connect to an OleTx transaction manager. The cbTmProtocolData field MUST be at least 2.
TmProtocolExtended 0x00000004	This field MUST contain an SExtendedEndpointInfo (section 2.2.5.8) structure for an extension protocol. The cbTmProtocolData field MUST be at least 16.

2.2.5.10 STxInfo

The STxInfo structure represents an exported transaction during push-based transaction propagation. The information in this structure is passed to a transaction manager in order to import a transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
guidSignature																															
...																															
...																															
...																															
uowTx (optional)																															
...																															
...																															
...																															
tmprotUsed (optional)																															
cbProtocolSpecificTxInfo (optional)																															
protocolSpecificTxInfo (variable)																															
...																															

guidSignature (16 bytes): This field MUST be a GUID that either specifies the transaction identifier or specifies a signature value that indicates that the fields following this field are present in the structure. If the field contains the binary value representation of the GUID {2adb4463-bd41-11d0-b12e-00c04fc2f3ef}, the fields **uowTx**, **tmprotUsed**, and **cbProtocolSpecificTxInfo** MUST be present. For all other values, this field MUST specify the GUID of the transaction to be imported, and all other fields MUST NOT be present.

uowTx (16 bytes): If present, this field MUST be a GUID that specifies the transaction identifier.

tmprotUsed (4 bytes): If present, this field MUST specify the format of the data in the **protocolSpecificTxInfo** field. The value MUST be one that is as specified in [TM PROTOCOL \(section 2.2.6.2\)](#).

cbProtocolSpecificTxInfo (4 bytes): If present, this field MUST contain the size of the protocol-specific data. This value MUST be zero, unless **tmprotUsed** contains the value TmProtocolTip, in which case the value MUST be 48.

protocolSpecificTxInfo (variable): If present, this field MUST contain a [SOleTxInfoForTip \(section 2.2.5.7\)](#) structure. If the **cbProtocolSpecificTxInfo** field is present and has a non-zero value, this field MUST be present. Otherwise, this field MUST not be present.

2.2.5.11 SWhereabouts

The SWhereabouts structure describes the location of a transaction manager and the protocols that MAY be used to contact it.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
guidSignature																															
...																															
...																															
cTmToTmProtocols																															
rgtmprotUsableList (variable)																															
...																															

guidSignature (16 bytes): This field contains a signature value for this structure. The value MUST be the binary representation of the GUID {2adb4462-bd41-11d0-b12e-00c04fc2f3ef}.

cTmToTmProtocols (4 bytes): This field MUST contain the number of [STmToTmProtocol \(section 2.2.5.9\)](#) structures present in the **rgtmprotUsableList** field. This value MUST be at least one.

rgtmprotUsableList (variable): This field MUST contain an unordered list of STmToTmProtocol structures with protocol-specific connection information for this transaction manager. Each entry MUST be aligned on a 4-byte boundary, and any necessary padding bytes MUST be set to an implementation-specific value, and MUST be ignored on receipt. [<15>](#)A list that contains an STmToTmProtocol with a **tmprotDescribed** value of TmProtocolMsdtcV2 MUST also contain an STmToTmProtocol with a **tmprotDescribed** value of TmProtocolMsdtcV1. In this case, the **wszHostName** value in the [SDtcCmEndpointV2](#) structure MUST be used in place of the **szHostName** value in the [SDtcCmEndpointV1](#) structure.

2.2.6 Transaction Enumerations

2.2.6.1 Connection Types

The CONNTYPE enumeration defines the **connection types** that are used by [MS-DTCO].

```
typedef enum
```

```

{
    CONNTYPE_TXUSER_BEGINNER = 0x00000001,
    CONNTYPE_TXUSER_IMPORT = 0x00000002,
    CONNTYPE_TXUSER_ENLISTMENT = 0x00000003,
    CONNTYPE_TXUSER_EXPORT = 0x00000004,
    CONNTYPE_TXUSER_RESOURCEMANAGER = 0x00000005,
    CONNTYPE_TXUSER_REENLIST = 0x00000006,
    CONNTYPE_TXUSER_RESOLVE = 0x00000007,
    CONNTYPE_TXUSER_VOTER = 0x00000009,
    CONNTYPE_TXUSER_ASSOCIATE = 0x00000011,
    CONNTYPE_TXUSER_GETTXDETAILS = 0x00000022,
    CONNTYPE_TXUSER_PHASE0 = 0x00000024,
    CONNTYPE_TXUSER_BEGIN2 = 0x00000028,
    CONNTYPE_TXUSER_IMPORT2 = 0x00000033,
    CONNTYPE_TXUSER_GETSECURITYFLAGS = 0x00000035,
    CONNTYPE_TXUSER_TRACE = 0x00000036,
    CONNTYPE_TXUSER_SETTXTIMEOUT = 0x00000037,
    CONNTYPE_TXUSER_SETTXTIMEOUT2 = 0x00000038,
    CONNTYPE_TXUSER_PROMOTE = 0x00000039,
    CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS = 0x0000003D,
    CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL = 0x00000046,
    CONNTYPE_PARTNERTM_PROPAGATE = 0x00000101,
    CONNTYPE_PARTNERTM_REDELIVERCOMMIT = 0x00000102,
    CONNTYPE_PARTNERTM_CHECKABORT = 0x00000103,
    CONNTYPE_PARTNERTM_BRANCH = 0x00000104
} CONNTYPE;

```

CONNTYPE_TXUSER_BEGINNER: This connection type is used by applications that begin, commit, and roll back transactions.

CONNTYPE_TXUSER_IMPORT: This connection type is used by a destination application to complete a [Push Propagation](#) that is initiated by a source application.

CONNTYPE_TXUSER_ENLISTMENT: This connection type is used by a durable resource manager to establish an enlistment with its transaction manager.

CONNTYPE_TXUSER_EXPORT: This connection type is used by a source application to initiate a Push Propagation to a destination application.

CONNTYPE_TXUSER_RESOURCEMANAGER: This connection type is used by a durable resource manager to register with its transaction manager.

CONNTYPE_TXUSER_REENLIST: This connection type is used by a durable resource manager to determine the outcome of an [In Doubt](#) transaction.

CONNTYPE_TXUSER_RESOLVE: This connection type is used by an application either to manually resolve the outcome of an In Doubt transaction or to cause its transaction manager to forget a transaction that is in the [Failed to Notify](#) state.

CONNTYPE_TXUSER_VOTER: This connection type is used by a volatile resource manager to establish a voter enlistment with its transaction manager.

CONNTYPE_TXUSER_ASSOCIATE: This connection type is used by a destination application to complete the [Pull Propagation](#) of a transaction from a source application.

CONNTYPE_TXUSER_GETTXDETAILS: This connection type is used by an application to retrieve details about a transaction from its transaction manager.

CONNTYPE_TXUSER_PHASE0: This connection type is used by a resource manager to enlist for [Phase Zero](#) notifications from its transaction manager.

CONNTYPE_TXUSER_BEGIN2: This connection type is used by an application to begin, commit, or roll back a transaction or to change the time-out of a transaction. This connection type supersedes [CONNTYPE_TXUSER_BEGINNER](#) and [CONNTYPE_TXUSER_SETTXTIMEOUT2](#).

CONNTYPE_TXUSER_IMPORT2: This connection type is used by a destination application to complete a Push Propagation that is initiated by a source application. This connection type supersedes [CONNTYPE_TXUSER_IMPORT](#).

CONNTYPE_TXUSER_GETSECURITYFLAGS: This connection type is used by an application to obtain the security configuration of its transaction manager.

CONNTYPE_TXUSER_TRACE: This connection type is used by an application to ask its transaction manager to trace the status of a transaction by using an implementation-specific mechanism.

CONNTYPE_TXUSER_SETTXTIMEOUT: This connection type is used by an application to modify the time-out of a transaction.

CONNTYPE_TXUSER_SETTXTIMEOUT2: This connection type is used by an application to modify the time-out of a transaction. This connection type supersedes [CONNTYPE_TXUSER_SETTXTIMEOUT](#).

CONNTYPE_TXUSER_PROMOTE: This connection type is used by an application to:

- Begin a transaction using an application-specified transaction identity
- Commit or **rollback** a transaction
- Change the time-out of a transaction

This connection type supersedes [CONNTYPE_TXUSER_SETTXTIMEOUT2](#).

CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS: This connection type is used by an application to obtain Extended Whereabouts from its transaction manager.

CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL: This connection type is used by a durable resource manager to register with a transaction manager and to detect duplicate registrations. This connection type supersedes [CONNTYPE_TXUSER_RESOURCEMANAGER](#).

CONNTYPE_PARTNERTM_PROPAGATE: This connection type is used by a superior transaction manager to do a Push Propagation of a transaction to its subordinate transaction manager and to execute the Two-Phase Commit Protocol.

CONNTYPE_PARTNERTM_REDELIVERCOMMIT: This connection type is used by a superior transaction manager to redeliver a Commit notification for a transaction to its subordinate transaction manager.

CONNTYPE_PARTNERTM_CHECKABORT: This connection type is used by a subordinate transaction manager to query the outcome of a transaction from its superior transaction manager.

CONNTYPE_PARTNERTM_BRANCH: A subordinate transaction manager uses this connection type to register a new subordinate enlistment with a superior transaction manager.

2.2.6.2 TM_Protocol

The **TM_PROTOCOL** enumeration defines types of transaction-manager-to-transaction-manager protocols that are available for use.

```
typedef enum
{
    TmProtocolNone = 0,
    TmProtocolTip = 1,
    TmProtocolMsdtcV1 = 2,
    TmProtocolMsdtcV2 = 3,
    TmProtocolExtended = 4
} TM_PROTOCOL;
```

TmProtocolNone: No transaction-manager-to-transaction-manager protocol is available.

TmProtocolTip: The Transaction Internet Protocol (TIP) protocol is available.

TmProtocolMsdtcV1: The OleTx protocol is available with basic information.

TmProtocolMsdtcV2: The OleTx protocol is available with additional information.

TmProtocolExtended: An extension protocol is available.

2.2.6.3 TXUSER_ENLISTMENT_PREPAREREQDONE_RESPONSE

The **TXUSER_ENLISTMENT_PREPAREREQDONE_RESPONSE** enumeration defines the status values for prepare request from a subordinate resource manager.

```
typedef enum
{
    TXUSER_ENLISTMENT_PREPAREREQDONE_OK = 0,
    TXUSER_ENLISTMENT_PREPAREREQDONE_ABORT = 1,
    TXUSER_ENLISTMENT_PREPAREREQDONE_READ_ONLY = 2,
    TXUSER_ENLISTMENT_PREPAREREQDONE_SINGLEPHASE_COMMIT = 3
} TXUSER_ENLISTMENT_PREPAREREQDONE_RESPONSE;
```

TXUSER_ENLISTMENT_PREPAREREQDONE_OK: The prepare request was successful, and the enlistment requires the transaction outcome.

TXUSER_ENLISTMENT_PREPAREREQDONE_ABORT: The prepare request was unsuccessful, and the transaction MUST be aborted.

TXUSER_ENLISTMENT_PREPAREREQDONE_READ_ONLY: The request to prepare the transaction for commitment was successful, and no further involvement in the transaction is required.

TXUSER_ENLISTMENT_PREPAREREQDONE_SINGLEPHASE_COMMIT: The sender chose the single-phase commit option and committed the transaction.

2.2.6.4 PARTNERTM_PROPAGATE_PREPAREREQDONE_RESPONSE

The **PARTNERTM_PROPAGATE_PREPAREREQDONE_RESPONSE** enumeration defines the status values for prepare request from a subordinate transaction manager.

```
typedef enum
{
    PARTNERTM_PROPAGATE_PREPAREREQDONE_OK = 0,
    PARTNERTM_PROPAGATE_PREPAREREQDONE_ABORT = 1,
    PARTNERTM_PROPAGATE_PREPAREREQDONE_READ_ONLY = 2,
    PARTNER_PROPAGATE_PREPAREREQDONE_SINGLEPHASE_COMMIT = 3
} PARTNERTM_PROPAGATE_PREPAREREQDONE_RESPONSE;
```

PARTNERTM_PROPAGATE_PREPAREREQDONE_OK: The prepare request was successful, and the enlistment requires the transaction outcome.

PARTNERTM_PROPAGATE_PREPAREREQDONE_ABORT: The prepare request was unsuccessful, and the transaction MUST be aborted.

PARTNERTM_PROPAGATE_PREPAREREQDONE_READ_ONLY: The request to prepare the transaction for commitment was successful, and no further involvement in the transaction is required.

PARTNER_PROPAGATE_PREPAREREQDONE_SINGLEPHASE_COMMIT: The sender chose the single-phase commit option and committed the transaction.

2.2.6.5 TXUSER_VOTER_VOTERREQDONE_RESPONSE

The **TXUSER_VOTER_VOTERREQDONE_RESPONSE** enumeration defines the status values for a prepare request from a subordinate resource manager.

```
typedef enum
{
    TXUSER_VOTER_VOTEREQDONE_OK = 0,
    TXUSER_VOTER_VOTEREQDONE_ABORT = 1,
    TXUSER_VOTER_VOTEREQDONE_READ_ONLY = 2
} TXUSER_VOTER_VOTERREQDONE_RESPONSE;
```

TXUSER_VOTER_VOTEREQDONE_OK: The prepare request was successful, and the voter requires the transaction outcome.

TXUSER_VOTER_VOTEREQDONE_ABORT: The prepare request was unsuccessful, and the transaction MUST be aborted.

TXUSER_VOTER_VOTEREQDONE_READ_ONLY: The prepare request was successful, and the voter does not require the transaction outcome.

2.2.6.6 TRUN_TXBEGIN_ERRORS

The **TRUN_TXBEGIN_ERRORS** enumeration defines the completion status values for requests from an application to perform the following steps in a transaction: begin, set time-out, commit, or abort a transaction.

```
typedef enum
{
    TRUN_TXBEGIN_ERROR_NO_MEM = 1,
    TRUN_TXBEGIN_ERROR_BEGIN_LOG_FULL = 20,
    TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED = 30,
    TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED = 31,
}
```

```

    TRUN_TXBEGIN_ERROR_NOTIFY_INDOUBT = 32,
    TRUN_TXBEGIN_ERROR_DUPLICATE_GUID = 33
} TRUN_TXBEGIN_ERRORS;

```

TRUN_TXBEGIN_ERROR_NO_MEM: There was insufficient memory to allocate the data structures necessary to create the new transaction.

TRUN_TXBEGIN_ERROR_BEGIN_LOG_FULL: There was insufficient space in the transaction manager log to accommodate a new transaction.

TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED: The transaction has aborted.

TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED: The transaction has committed.

TRUN_TXBEGIN_ERROR_NOTIFY_INDOUBT: The transaction has completed, but the outcome is no longer determinable. This occurs if the transaction manager delegated the commit decision to a subordinate through the single-phase commit protocol and if the connection to that subordinate terminated before the result could be reported.

TRUN_TXBEGIN_ERROR_DUPLICATE_GUID: An attempt was made to promote a transaction, but a transaction with the specified Transaction Identifier already exists.

2.2.6.7 TRUN_TXIMPORT_ERRORS

The **TRUN_TXIMPORT_ERRORS** enumeration defines the completion status values for requests to import a transaction or to abort a transaction that was previously imported.

```

typedef enum
{
    TRUN_TXIMPORT_ERROR_NO_MEM = 1,
    TRUN_TXIMPORT_ERROR_IMPORT_TX_NOT_FOUND = 20,
    TRUN_TXIMPORT_ERROR_NOTIFY_ABORTED = 30,
    TRUN_TXIMPORT_ERROR_NOTIFY_COMMITTED = 31,
    TRUN_TXIMPORT_ERROR_NOTIFY_INDOUBT = 32
} TRUN_TXIMPORT_ERRORS;

```

TRUN_TXIMPORT_ERROR_NO_MEM: There was not enough memory to complete the operation.

TRUN_TXIMPORT_ERROR_IMPORT_TX_NOT_FOUND: The specified transaction was not found.

TRUN_TXIMPORT_ERROR_NOTIFY_ABORTED: The transaction aborted.

TRUN_TXIMPORT_ERROR_NOTIFY_COMMITTED: The transaction committed.

TRUN_TXIMPORT_ERROR_NOTIFY_INDOUBT: The transaction completed, but the outcome could not be determined.

2.2.6.8 OLETX_ISOLATION_FLAGS

The **OLETX_ISOLATION_FLAGS** bitfield enumeration values specify isolation flags for a transaction. [<16>](#)

```
typedef enum
{
    ISOFLAG_RETAIN_COMMIT_DC = 0x00000001,
    ISOFLAG_RETAIN_COMMIT = 0x00000002,
    ISOFLAG_RETAIN_COMMIT_NO = 0x00000003,
    ISOFLAG_RETAIN_ABORT_DC = 0x00000004,
    ISOFLAG_RETAIN_ABORT = 0x00000008,
    ISOFLAG_RETAIN_ABORT_NO = 0x0000000C,
    ISOFLAG_RETAIN_DONTCARE = 0x00000005,
    ISOFLAG_RETAIN_BOTH = 0x0000000A,
    ISOFLAG_RETAIN_NONE = 0x0000000F,
    ISOFLAG_RETAIN_DEFAULT = 0x00000000,
    ISOFLAG_OPTIMISTIC = 0x00000010,
    ISOFLAG_READONLY = 0x00000020
} OLETX_ISOLATION_FLAGS;
```

ISOFLAG_RETAIN_COMMIT_DC: Retain locks on transaction commit, regardless of the success or failure of that commit request.

If this value is set, then ISOFLAG_RETAIN_COMMIT and ISOFLAG_RETAIN_COMMIT_NO MUST NOT be set.

ISOFLAG_RETAIN_COMMIT: Retain locks on a successful transaction commit.

If this value is set, then ISOFLAG_RETAIN_COMMIT_DC and ISOFLAG_RETAIN_COMMIT_NO MUST NOT be set.

ISOFLAG_RETAIN_COMMIT_NO: Do not retain locks on a transaction commit.

If this value is set, then ISOFLAG_RETAIN_COMMIT_DC and ISOFLAG_RETAIN_COMMIT MUST NOT be set.

ISOFLAG_RETAIN_ABORT_DC: Retain locks on transaction abort, regardless of the success or failure of that Abort request.

If this value is set, then ISOFLAG_RETAIN_ABORT and ISOFLAG_RETAIN_ABORT_NO MUST NOT be set.

ISOFLAG_RETAIN_ABORT: Retain locks on a successful transaction abort.

If this value is set, then ISOFLAG_RETAIN_ABORT_DC and ISOFLAG_RETAIN_ABORT_NO MUST NOT be set.

ISOFLAG_RETAIN_ABORT_NO: Do not retain locks on a transaction abort.

If this value is set, then ISOFLAG_RETAIN_ABORT and ISOFLAG_RETAIN_ABORT_DC MUST NOT be set.

ISOFLAG_RETAIN_DONTCARE: Retain locks on all transaction termination requests, regardless whether the request was to abort or commit.

This is a synonym for selecting ISOFLAG_RETAIN_COMMIT_DC and ISOFLAG_RETAIN_ABORT_DC.

ISOFLAG_RETAIN_BOTH: Retain locks on all successful transaction termination requests, regardless of whether or not the request was to abort or commit.

This is a synonym for selecting ISOFLAG_RETAIN_COMMIT and ISOFLAG_RETAIN_ABORT.

ISOFLAG_RETAIN_NONE: Do not retain locks on any transaction termination requests.

This is a synonym for selecting ISOFLAG_RETAIN_COMMIT_NO and ISOFLAG_RETAIN_ABORT_NO.

ISOFLAG_RETAIN_DEFAULT: Do not retain locks on any transaction termination requests.

This is a synonym for selecting ISOFLAG_RETAIN_NONE.

ISOFLAG_OPTIMISTIC: Optimistic locking is allowed.

ISOFLAG_READONLY: The transaction is not expected to modify data.

2.2.6.9 OLETX_ISOLATION_LEVEL

The **OLETX_ISOLATION_LEVEL** enumeration values specify the isolation levels of a transaction.

```
typedef enum
{
    ISOLATIONLEVEL_UNSPECIFIED = 0xffffffff,
    ISOLATIONLEVEL_CHAOS = 0x10,
    ISOLATIONLEVEL_READUNCOMMITTED = 0x100,
    ISOLATIONLEVEL_READCOMMITTED = 0x1000,
    ISOLATIONLEVEL_REPEATABLE_READ = 0x10000,
    ISOLATIONLEVEL_SERIALIZABLE = 0x100000
} OLETX_ISOLATION_LEVEL;
```

ISOLATIONLEVEL_UNSPECIFIED: No isolation level was specified.

ISOLATIONLEVEL_CHAOS: Data is not isolated.

ISOLATIONLEVEL_READUNCOMMITTED: A transaction can read any data, even if it is being modified by another transaction. Any type of new data can be inserted during a transaction.

ISOLATIONLEVEL_READCOMMITTED: A transaction MUST NOT read data that is being modified by another transaction that has not committed. Any type of new data can be inserted during a transaction.

ISOLATIONLEVEL_REPEATABLE_READ: Data read by a current transaction MUST NOT be changed by another transaction until the current transaction finishes. Any type of new data can be inserted during a transaction.

ISOLATIONLEVEL_SERIALIZABLE: Data read by a current transaction MUST NOT be changed by another transaction until the current transaction finishes. New data MUST NOT be inserted by another transaction that would affect the current transaction.

2.2.7 Transaction Constants

2.2.7.1 GRFRM

This MUST be a 32-bit unsigned integer that contains an implementation-defined value. This value MAY be ignored on receipt. [<17>](#)

2.2.7.2 DTCADVCONFIG

These flags indicate the remote communications that are enabled for a transaction manager protocol.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	B	C	D	E	F	G	H	I	J	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Marker Bits

Value	Description
A	DTCADVCONFIG_NETWORKDTCACCESS_ENABLE This bit corresponds to the Allow Network Access flag maintained by the Core Transaction Manager Facet (section 1.3.3.3.1) , as defined in Core Transaction Manager Facet Details (section 3.2) .
B	DTCADVCONFIG_NETWORKDTCACCESS_ADMIN This bit corresponds to the Allow Remote Administration flag maintained by the Core Transaction Manager Facet (section 1.3.3.3.1), as defined in Core Transaction Manager Facet Details (section 3.2).
C	DTCADVCONFIG_NETWORKDTCACCESS_TX This bit corresponds to the Allow Network Transactions flag maintained by the Core Transaction Manager Facet (section 1.3.3.3.1), as defined in Core Transaction Manager Facet Details (section 3.2).
D	DTCADVCONFIG_NETWORKDTCACCESS_CLIENTS This bit corresponds to the Allow Network Clients flag maintained by the Core Transaction Manager Facet (section 1.3.3.3.1), as defined in Core Transaction Manager Facet Details (section 3.2).
E	DTCADVCONFIG_NETWORKDTCACCESS_TIP This bit corresponds to the Allow TIP flag maintained by the Core Transaction Manager Facet (section 1.3.3.3.1) as defined in Core Transaction Manager Facet Details (section 3.2).
F	DTCADVCONFIG_OUTBOUNDNETWORK_TX This bit corresponds to the Allow Outbound Transactions flag maintained by the Core Transaction Manager Facet (section 1.3.3.3.1) as defined in Core Transaction Manager Facet Details (section 3.2).
G	DTCADVCONFIG_INBOUNDNETWORK_TX This bit corresponds to the Allow Inbound Transactions flag maintained by the Core Transaction Manager Facet (section 1.3.3.3.1) as defined in Core Transaction Manager Facet Details (section 3.2).
H	DTCADVCONFIG_SECURITYLEVEL_NOSECURITY This bit MUST be ignored if either DTCADVCONFIG_SECURITYLEVEL_AUTHENTICATEDONLY and DTCADVCONFIG_SECURITYLEVEL_MUTUALAUTH are set <18> . Otherwise setting this bit corresponds to the No Security value of the Security Level enumeration maintained by the Core Transaction Manager Facet (section 1.3.3.3.1) as defined in Core Transaction

Value	Description
	Manager Facet Details (section 3.2).
I	DTCADVCONFIG_SECURITYLEVEL_AUTHENTICATEDONLY This bit MUST be ignored if DTCADVCONFIG_SECURITYLEVEL_MUTUALAUTH is set. Otherwise setting this bit corresponds to the Incoming Authentication value of the Security Level enumeration maintained by the Core Transaction Manager Facet (section 1.3.3.3.1) as defined in Core Transaction Manager Facet Details (section 3.2).
J	DTCADVCONFIG_SECURITYLEVEL_MUTUALAUTH This bit corresponds to the Mutual Authentication value of the Security Level enumeration maintained by the Core Transaction Manager Facet (section 1.3.3.3.1) as defined in Core Transaction Manager Facet Details (section 3.2).
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.
X	SHOULD be set to zero, and MUST be ignored when read.

2.2.8 Connection Types Relevant to Applications

2.2.8.1 Transaction Initiation and Completion

2.2.8.1.1 CONNTYPE_TXUSER_BEGINNER

This connection type is used by applications that begin, commit, and roll back transactions.

For more information about CONNTYPE_TXUSER_BEGINNER as an initiator, see section [3.3.5.1.1](#), and as an acceptor, see section [3.4.5.1.1](#).

2.2.8.1.1.1 TXUSER_BEGINNER_MTAG_ABORT

This message requests an attempt to abort the transaction that was begun on this connection.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
guidReason																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001013.
- The **dwcbVarLenData** field MUST be 16.

guidReason (16 bytes): The value MUST be set to an implementation-specific value and SHOULD be ignored on receipt. [<19>](#)

2.2.8.1.1.2 TXUSER_BEGINNER_MTAG_BEGIN

This message requests the creation of a transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
isoLevel																															
dwTimeout																															
szDesc																															
...																															
...																															
...																															
...																															
...																															
...																															
(szDesc cont'd for 2 rows)																															
isoFlags																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001011.
- The **dwcbVarLenData** field MUST be 52.

isoLevel (4 bytes): The [OLETX ISOLATION LEVEL](#) enumeration that is defined for the transaction.<20>

dwTimeout (4 bytes): A 32-bit unsigned integer that MUST contain the time-out value, in milliseconds, for the transaction. The value zero MUST be interpreted as an infinite time-out. A transaction SHOULD NOT abort due to time-out before the time-out that is specified by this value has expired.

szDesc (40 bytes): The description of the transaction, as a fixed-size array of 40 bytes that contains a null-terminated Latin-1 ANSI string, as specified in [\[ISO-8859-1\]](#). See section [2.2.5.4](#) for details.

isoFlags (4 bytes): The [OLETX ISOLATION FLAGS](#) enumeration that is defined for the transaction.<21>

2.2.8.1.1.3 TXUSER_BEGINNER_MTAG_BEGIN_LOG_FULL

This message indicates the transaction was not created because the transaction recovery log had insufficient space to accommodate the new transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001018.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.1.1.4 TXUSER_BEGINNER_MTAG_BEGIN_NO_MEM

This message indicates the transaction was not created because of insufficient memory.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

- MsgHeader (24 bytes):** This field MUST contain a [MESSAGE_PACKET](#) structure.
- The **dwUserMsgType** field MUST be 0x00001019.
 - The **dwcbVarLenData** field MUST be 0.

2.2.8.1.1.5 TXUSER_BEGINNER_MTAG_BEGUN

This message indicates the request to begin a transaction was successful.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001012.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier. The value MUST NOT be set to a **NULL GUID**.

2.2.8.1.1.6 TXUSER_BEGINNER_MTAG_COMMIT

This message requests an attempt to commit the transaction that was begun on this connection.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
grfRM																															
fAsyncFull																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001014.
- The **dwcbVarLenData** field MUST be 8.

grfRM (4 bytes): The value of this field MUST be as specified in [GRFRM \(section 2.2.7.1\)](#).

fAsyncFull (4 bytes): Reserved. This value MUST be set to zero and MUST be ignored on receipt.

2.2.8.1.1.7 TXUSER_BEGINNER_MTAG_COMMIT_INDOUBT

This message indicates that the transaction manager is unable to determine, and will never be able to determine, the outcome of the transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001990.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.1.1.8 TXUSER_BEGINNER_MTAG_COMMIT_TOO_LATE

This message indicates that the commit request cannot be completed successfully because it is too late in the lifetime of the transaction to commit it.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001016.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.1.1.9 TXUSER_BEGINNER_MTAG_REQUEST_COMPLETED

This message is sent to indicate the request was completed successfully.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001015.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.1.2 CONNTYPE_TXUSER_BEGIN2

This connection type is used by an application to begin, commit, or roll back a transaction or to change the time-out of a transaction. This connection type supersedes [CONNTYPE_TXUSER_BEGINNER](#) and [CONNTYPE_TXUSER_SETTXTIMEOUT2](#).

For more information about CONNTYPE_TXUSER_BEGIN2 as an initiator, see [3.3.5.1.2](#), and as an acceptor, see [3.4.5.1.2](#).

2.2.8.1.2.1 TXUSER_BEGIN2_MTAG_ABORT

The TXUSER_BEGIN2_MTAG_ABORT message requests an attempt to abort the transaction that was begun on this connection.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00006001.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.1.2.2 TXUSER_BEGIN2_MTAG_BEGIN

This message is used to request the creation of a transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
isoLevel																															
dwTimeout																															

szDesc
...
...
...
...
...
...
...
...
(szDesc cont'd for 2 rows)
isoFlags

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure:

- The **dwUserMsgType** field MUST be 0x00006002.
- The **dwcbVarLenData** field MUST be 52.

isoLevel (4 bytes): See the **isoLevel** field in section [2.2.8.1.1.2](#) for details.

dwTimeout (4 bytes): See the **dwTimeout** field in section [2.2.8.1.1.2](#) for details.

szDesc (40 bytes): See the **szDesc** field in section [2.2.8.1.1.2](#) for details.

isoFlags (4 bytes): See the **isoFlags** field in section [2.2.8.1.1.2](#) for details.

2.2.8.1.2.3 TXUSER_BEGIN2_MTAG_COMMIT

The TXUSER_BEGIN2_MTAG_COMMIT message requests an attempt to commit the transaction that was begun on this connection.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
MsgHeader																															
...																															
...																															
...																															
...																															
grfRM																															

- MsgHeader (24 bytes):** This field MUST contain a [MESSAGE_PACKET](#) structure:
- The **dwUserMsgType** field MUST be 0x00006003.
 - The **dwcbVarLenData** field MUST be 4.
- grfRM (4 bytes):** The value of this field MUST be as specified in [GRFRM](#).

2.2.8.1.2.4 TXUSER_BEGIN2_MTAG_SINK_BEGUN

This message indicates the request to begin a transaction was successful.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure:

- The **dwUserMsgType** field MUST be 0x00006006.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier. This value MUST NOT be a NULL GUID.

2.2.8.1.2.5 TXUSER_BEGIN2_MTAG_SINK_ERROR

The content of this message provides information about the outcome of a request.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
Error																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure:

- The **dwUserMsgType** field MUST be 0x00006005.
- The **dwcbVarLenData** field MUST be 4.

Error (4 bytes): This field MUST contain the status for the previous request. The value MUST be a member of the [TRUN_TXBEGIN_ERRORS](#) enumeration.

2.2.8.1.2.6 TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE

This message indicates that the transaction time-out was successfully modified.

This message is also used for [CONNTYPE_TXUSER_SETTXTIMEOUT](#).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure:

- The **dwUserMsgType** field MUST be 0x0000107C.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.1.2.7 TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT

This message modifies the transaction time-out when it is used in [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) and [CONNTYPE_TXUSER_SETTXTIMEOUT \(section 2.2.8.3.3\)](#), or queries if the transaction manager supports the capability to do so when used in [CONNTYPE_TXUSER_SETTXTIMEOUT2 \(section 2.2.8.3.4\)](#).

This message is also used for CONNTYPE_TXUSER_SETTXTIMEOUT (section 2.2.8.3.3) and CONNTYPE_TXUSER_SETTXTIMEOUT2 (section 2.2.8.3.4).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															
dwTxTimeout																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure:

- The **dwUserMsgType** field MUST be 0x0000107B.
- The **dwcbVarLenData** field MUST be 20.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier. When this message is sent on a CONNTYPE_TXUSER_SETTXTIMEOUT2 (section 2.2.8.3.4) connection to query the capability of the transaction manager, this value SHOULD be set to a NULL GUID, and MUST be ignored on receipt.

dwTxTimeout (4 bytes): A 32-bit unsigned integer that contains the new time-out value, in milliseconds, for the transaction. When used with a CONNTYPE_TXUSER_BEGIN2 (section 2.2.8.1.2) connection, a transaction MUST NOT abort due to time-out before the number of milliseconds that is specified by the value has expired. The value zero MUST be interpreted as an infinite time-out. When used with a CONNTYPE_TXUSER_SETTXTIMEOUT2 (section 2.2.8.3.4) connection, this value SHOULD be set to zero and MUST be ignored on receipt.

2.2.8.1.2.8 TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE

This message indicates that it is too late to modify the time-out of the transaction.

This message is also in the [CONNTYPE_TXUSER_SETTXTIMEOUT](#) and [CONNTYPE_TXUSER_SETTXTIMEOUT2](#) connection types.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure:

- The **dwUserMsgType** field MUST be 0x0000107E.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.1.3 CONNTYPE_TXUSER_PROMOTE

This connection type is used by an application to do the following:

- Begin a transaction using an application-specified transaction identity.
- Commit or roll back a transaction.
- Change the time-out of a transaction.

This connection type supersedes [CONNTYPE_TXUSER_SETTXTIMEOUT2](#).

For more information about CONNTYPE_TXUSER_PROMOTE as an initiator, see section [3.3.5.1.3](#), and as an acceptor, see section [3.4.5.1.3](#).

2.2.8.1.3.1 TXUSER_BEGINNER_MTAG_PROMOTE

This message is used to request the creation of a transaction that specifies a predetermined transaction identifier.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															

...
...
...
...
...
isoLevel
dwTimeout
szDesc
...
...
...
...
...
...
...
...
(szDesc cont'd for 2 rows)
isoFlags
guidTx
...
...
...

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure:

- The **dwUserMsgType** field MUST be 0x00001010.
- The **dwcbVarLenData** field MUST be 68.

isoLevel (4 bytes): See the **isoLevel** field in section [2.2.8.1.1.2](#) for details.

dwTimeout (4 bytes): See the **dwTimeout** field in section [2.2.8.1.1.2](#) for details.

szDesc (40 bytes): See the **szDesc** field in section [2.2.8.1.1.2](#) for details.

isoFlags (4 bytes): See the **isoFlags** field in section [2.2.8.1.1.2](#) for details.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.8.2 Transaction Propagation

2.2.8.2.1 Pull Propagation

2.2.8.2.1.1 CONNTYPE_TXUSER_ASSOCIATE

This connection type is used by a destination application to complete the [Pull Propagation](#) of a transaction from a source application.

For more information about CONNTYPE_TXUSER_ASSOCIATE as an initiator, see section [3.3.5.2.1.1](#), and as an acceptor, see section [3.4.5.2.1.1](#).

2.2.8.2.1.1.1 TXUSER_ASSOCIATE_MTAG_ASSOCIATE

This message requests that the transaction manager perform pull-based propagation of an existing transaction. This is also known as an associate request.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTx																															

...
...
...
isoLevel
isoFlags
cbSourceTmAddr
szDesc
...
...
...
...
...
...
...
...
...
(szDesc cont'd for 2 rows)
SourceTmAddr (variable)
...

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002031.
- The **dwcbVarLenData** field MUST be equal to the value of **cbSourceTmAddr** plus 68.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

isoLevel (4 bytes): See the **isoLevel** field in section [2.2.8.1.1.2](#) for details.

isoFlags (4 bytes): See the **isoFlags** field in section [2.2.8.1.1.2](#) for details.

cbSourceTmAddr (4 bytes): A 4-byte integer value that MUST contain the length, in bytes, of the **SourceTmAddr** member.

szDesc (40 bytes): See the **szDesc** field in section [2.2.8.1.1.2](#) for details.

SourceTmAddr (variable): This field is used for identifying the address of the superior transaction manager against which the pull propagation operation is requested. This field MUST contain either a [NAMEOBJECTBLOB \(section 2.2.5.3\)](#) structure or a [OLETX_TM_ADDR \(section 2.2.4.2\)](#) structure in a version-specific manner as specified in section [Connection Types Relevant to Applications \(section 2.2.8\)](#).

2.2.8.2.1.1.2 TXUSER_ASSOCIATE_MTAG_ASSOCIATED

This message indicates that the associate request was successful.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002032.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.1.1.3 TXUSER_ASSOCIATE_MTAG_COMM_FAILED

This message indicates the associated request failed because the sender of this message encountered a communication failure with the Superior Transaction Manager specified in the **SourceTmAddr** field of the [TXUSER_ASSOCIATE_MTAG_ASSOCIATE \(section 2.2.8.2.1.1.1\)](#) message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002034.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.1.1.4 TXUSER_ASSOCIATE_MTAG_CREATE_BAD_TMADDR

This message indicates the associate request failed because of failures during interpretation and processing of the **SourceTmAddr** field in the [TXUSER_ASSOCIATE_MTAG_ASSOCIATE \(section 2.2.8.2.1.1.1\)](#) message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002044.

- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.1.1.5 TXUSER_ASSOCIATE_MTAG_LOG_FULL_LOCAL

This message indicates that the associate request failed because the transaction recovery log was full at the transaction manager sending this message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002035.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.1.1.6 TXUSER_ASSOCIATE_MTAG_LOG_FULL_REMOTE

This message indicates first that the associated request failed because of a full transaction recovery log at the superior transaction manager specified in the **SourceTmAddr** field of the [TXUSER_ASSOCIATE_MTAG_ASSOCIATE \(section 2.2.8.2.1.1.1\)](#) message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002037.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.1.1.7 TXUSER_ASSOCIATE_MTAG_NO_MEM_LOCAL

This message indicates that the associate request failed because of a failure to allocate dynamic memory by the transaction manager sending this message while processing the [TXUSER_ASSOCIATE_MTAG_ASSOCIATE \(section 2.2.8.2.1.1.1\)](#) message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002036.

- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.1.1.8 TXUSER_ASSOCIATE_MTAG_NO_MEM_REMOTE

This message indicates that the associate request failed because of a failure to allocate dynamic memory by the Superior Transaction Manager specified in the **SourceTmAddr** field of the [TXUSER_ASSOCIATE_MTAG_ASSOCIATE \(section 2.2.8.2.1.1.1\)](#) message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002038.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.1.1.9 TXUSER_ASSOCIATE_MTAG_TOO_LATE

This message is sent in response to a [TXUSER_ASSOCIATE_MTAG_ASSOCIATE \(section 2.2.8.2.1.1.1\)](#) message. It indicates that the associate request failed because the transaction specified by the **guidTx** field in the TXUSER_ASSOCIATE_MTAG_ASSOCIATE message is neither in the Active nor Phase Zero state.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002040.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.1.1.10 TXUSER_ASSOCIATE_MTAG_TOO_MANY_LOCAL

This message indicates that the associate request failed because the number of direct participants for the transaction specified by the guidTx field in the [TXUSER_ASSOCIATE_MTAG_ASSOCIATE \(section 2.2.8.2.1.1.1\)](#) message exceeded an implementation-specific limit by the transaction manager sending this message while processing the TXUSER_ASSOCIATE_MTAG_ASSOCIATE message. [<22>](#)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002041.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.1.1.11 TXUSER_ASSOCIATE_MTAG_TOO_MANY_REMOTE

This message indicates a failure by the associate request. The number of direct participants for the transaction that is specified by the **guidTx** field in the [TXUSER_ASSOCIATE_MTAG_ASSOCIATE \(section 2.2.8.2.1.1.1\)](#) message exceeded an implementation-specific limit at the Superior Transaction Manager. This limit is specified in the **SourceTmAddr** field of the TXUSER_ASSOCIATE_MTAG_ASSOCIATE message. [<23>](#)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002042.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.1.1.12 TXUSER_ASSOCIATE_MTAG_TX_NOT_FOUND

This message indicates that the associate request failed because the transaction specified by the **guidTx** field in the [TXUSER_ASSOCIATE_MTAG_ASSOCIATE \(section 2.2.8.2.1.1.1\)](#) message was not found.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002043.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2 Push Propagation

2.2.8.2.2.1 CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS

This connection type is used by an application to obtain Extended Whereabouts from its transaction manager.

For more information about CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS as an initiator, see section [3.3.5.2.2.1](#), and as an acceptor, see section [3.4.5.2.2.1](#).

2.2.8.2.2.1.1 TXUSER_EXTENDEDWHEREABOUTS_MTAG_GET

This message is sent by the application to the transaction manager to obtain the Extended Whereabouts of the transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00005A01.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.1.2 TXUSER_EXTENDEDWHEREABOUTS_MTAG_GOT

This message returns the set of extended whereabouts elements.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
dwProtocolCount																															
rgtmprotUsableList (variable)																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00005A02.
- The **dwcbVarLenData** field MUST be the number of bytes used by the **rgtmprotUsableList** field plus 4.

dwProtocolCount (4 bytes): An unsigned 32-bit value that MUST contain the number of elements in the rgtmprotUsableList array. If this value is zero, the **rgtmprotUsableList** field MUST be omitted.

rgtmprotUsableList (variable): Array of [STmToTmProtocol \(section 2.2.5.9\)](#) elements, each of which MUST be of type TmProtocolExtended. Each element defines the location information for an extension protocol. Each element MUST be aligned on a 4-byte boundary.

2.2.8.2.2.1.3 TXUSER_EXTENDEDWHEREABOUTS_MTAG_NOMEM

This message is sent by the transaction manager to the **Client** in response to a [TXUSER_EXTENDEDWHEREABOUTS_MTAG_GET \(section 2.2.8.2.2.1.1\)](#) message to indicate that there is not enough memory to process the request.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00005A03.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.2 CONNTYPE_TXUSER_EXPORT

This connection type is used by a source application to initiate a [Push Propagation](#) to a destination application.

For more information about CONNTYPE_TXUSER_EXPORT as an initiator, see section [3.3.5.2.2.2](#), and as an acceptor, see section [3.4.5.2.2.2](#).

2.2.8.2.2.2.1 TXUSER_EXPORT_MTAG_CREATE

This message is used by applications to establish a connection with the transaction manager in order to export transactions to a destination transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
SourceTmAddr (variable)																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001041.
- The **dwcbVarLenData** field MUST be the number of bytes used for the **SourceTmAddr** field, and the value MUST be at least 40.

SourceTmAddr (variable): This field MUST contain an [OLETX_TM_ADDR](#) structure that specifies the network address and identification information for the destination transaction manager. This transaction manager receives push propagation operations from the source transaction manager, which is the recipient of this message.
[CONNTYPE_PARTNERTM_PROPAGATE \(section 2.2.9.1.1.1\)](#) defines the protocol that is used between the two transaction managers as a result of the export operation.

2.2.8.2.2.2.2 TXUSER_EXPORT_MTAG_CREATE2

This message is used by applications to establish a connection with the transaction manager in preparation to export transactions to a destination transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
SourceTmAddr (variable)																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001804.
- The **dwcbVarLenData** field MUST be the number of bytes used for the **SourceTmAddr** field, and the value MUST be at least 40.

SourceTmAddr (variable): See the **SourceTmAddr** field in section [2.2.8.2.2.1](#) for details.

2.2.8.2.2.2.3 TXUSER_EXPORT_MTAG_CREATE_BAD_TMADDR

This message indicates that the create request failed because of errors encountered during the interpretation and processing of the **SourceTmAddr** field in the connection's initial [TXUSER_EXPORT_MTAG_CREATE](#) (section [2.2.8.2.2.2.1](#)) or [TXUSER_EXPORT_MTAG_CREATE2](#) (section [2.2.8.2.2.2.2](#)) message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001046.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.2.4 TXUSER_EXPORT_MTAG_CREATE_NET_TX_DISABLED

This message indicates that the create request failed because the transaction manager that received the request has disabled the ability to export transactions to other transaction managers. See the Allow Outbound Transaction flag in [Abstract Data Model \(section 3.2.1\)](#) for more details.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001805.

- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.2.5 TXUSER_EXPORT_MTAG_CREATED

This message indicates that the create request succeeded and the connection is now ready to process export requests.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001042.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.2.6 TXUSER_EXPORT_MTAG_EXPORT

This message is used to export a transaction to the destination transaction manager that is identified by **SourceTmAddr** in the connection's initial [TXUSER_EXPORT_MTAG_CREATE](#) (section 2.2.8.2.2.2.1) or [TXUSER_EXPORT_MTAG_CREATE2](#) (section 2.2.8.2.2.2.2) message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTX																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001043.
- The **dwcbVarLenData** field MUST be 16.

guidTX (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.8.2.2.2.7 TXUSER_EXPORT_MTAG_EXPORT_LOG_FULL

This message indicates that the export request failed because the transaction recovery log was full at the source transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001050.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.8 TXUSER_EXPORT_MTAG_EXPORT_NO_MEM

This message indicates the export request failed because the source transaction manager was unable to allocate sufficient dynamic memory for the request.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001802.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.2.9 TXUSER_EXPORT_MTAG_EXPORT_TOO_LATE

This message indicates the export request failed because it was too late to process the export request for the current state of the transaction. See [Export Transaction \(section 3.2.7.21\)](#) and [Export Transaction Failure \(section 3.4.7.11\)](#) for more information.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001049.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.2.10 TXUSER_EXPORT_MTAG_EXPORT_TOO_MANY

This message indicates that the export request failed because the number of direct participants for the transaction specified by the **guidTX** field in the [TXUSER_EXPORT_MTAG_EXPORT \(section 2.2.8.2.2.2.6\)](#) message exceeded the implementation-specific limit imposed by the source transaction manager.<24>

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001801.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.11 TXUSER_EXPORT_MTAG_EXPORT_TX_NOT_FOUND

This message indicates that the export request failed because the transaction specified by the **guidTX** field in the [TXUSER_EXPORT_MTAG_EXPORT \(section 2.2.8.2.2.6\)](#) message was not found by the source transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001048.

- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.2.12 TXUSER_EXPORT_MTAG_EXPORTED

This message indicates that the export request was successful.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001044.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.3 CONNTYPE_TXUSER_IMPORT

This connection type is used by a destination application to complete a [Push Propagation](#) that is initiated by a source application.

For more information about CONNTYPE_TXUSER_IMPORT as an initiator, see section [3.3.5.2.2.3](#), and as an acceptor, see section [3.4.5.2.2.3](#).

2.2.8.2.2.3.1 TXUSER_IMPORT_MTAG_ABORT

This message is a request for the transaction manager to abort the transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidReason																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001023.
- The **dwcbVarLenData** field MUST be 16.

guidReason (16 bytes): The value MUST be set to an implementation-specific value and SHOULD be ignored on receipt.[<25>](#)

2.2.8.2.2.3.2 TXUSER_IMPORT_MTAG_ABORT_TOO_LATE

This message is sent to the application in the connection type that was created for the originating [TXUSER_IMPORT_MTAG_IMPORT \(section 2.2.8.2.2.3.3\)](#) message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001025.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.3.3 TXUSER_IMPORT_MTAG_IMPORT

This message is used by a destination application to complete a [push propagation](#) operation that is initiated by a source application.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001021.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.8.2.2.3.4 TXUSER_IMPORT_MTAG_IMPORT_TX_NOT_FOUND

This message is sent if the attempt to import the transaction is unsuccessful because the transaction that is specified in the [TXUSER_IMPORT_MTAG_IMPORT \(section 2.2.8.2.2.3.3\)](#) message cannot be found.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

- MsgHeader (24 bytes):** This field MUST contain a [MESSAGE_PACKET](#) structure.
- The **dwUserMsgType** field MUST be 0x00001026.
 - The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.3.5 TXUSER_IMPORT_MTAG_IMPORTED

This message indicates that the import operation completed successfully.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
isoLevel																															
isoFlags																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

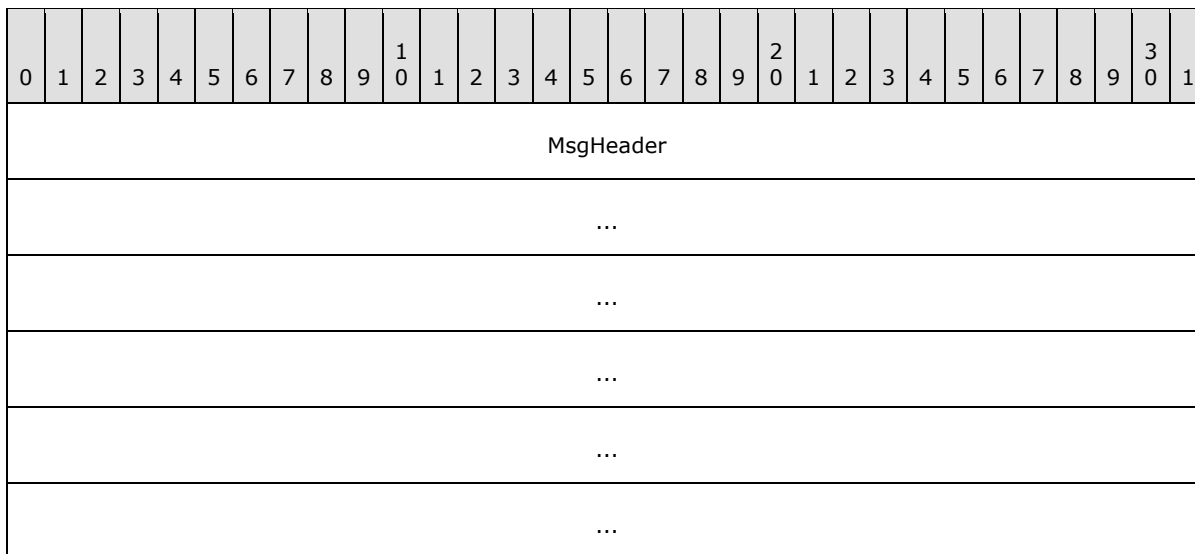
- The **dwUserMsgType** field MUST be 0x00001022.
- The **dwcbVarLenData** field MUST be 8.

isoLevel (4 bytes): See the **isoLevel** field in section [2.2.8.1.1.2](#) for details.

isoFlags (4 bytes): See the **isoFlags** field in section [2.2.8.1.1.2](#) for details.

2.2.8.2.2.3.6 TXUSER_IMPORT_MTAG_REQUEST_COMPLETED

This message indicates that the attempt to abort the transaction was successful.



MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001024.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.4 CONNTYPE_TXUSER_IMPORT2

This connection type is used by a destination application to complete a [Push Propagation](#) that is initiated by a source application. This connection type supersedes [CONNTYPE_TXUSER_IMPORT](#).

For more information about CONNTYPE_TXUSER_IMPORT2 as an initiator, see section [3.3.5.2.2.4](#), and as an acceptor, see section [3.4.5.2.2.4](#).

2.2.8.2.2.4.1 TXUSER_IMPORT2_MTAG_ABORT

This message is used to abort a transaction that was previously successfully imported by using either the [TXUSER_IMPORT2_MTAG_IMPORT \(section 2.2.8.2.2.4.2\)](#) or [TXUSER_IMPORT2_MTAG_IMPORT_WITH_SET \(section 2.2.8.2.2.4.3\)](#) message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00006101.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.2.2.4.2 TXUSER_IMPORT2_MTAG_IMPORT

This message is used by resource managers or **server** processes to establish a transaction connection with their transaction manager. The transaction identifier for which the connection is wanted is identified by the **guidTX** member of the message structure.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
guidTX																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00006102.
- The **dwcbVarLenData** field MUST be 16.

guidTX (16 bytes): This field MUST be a GUID that specifies the transaction identifier.

2.2.8.2.2.4.3 TXUSER_IMPORT2_MTAG_IMPORT_WITH_SET

This message is used by a destination application to complete a [push propagation](#) operation that is initiated by a source application. It is similar to the [TXUSER_IMPORT2_MTAG_IMPORT \(section 2.2.8.2.2.4.2\)](#) message, except that it allows the application to specify the isolation level, isolation flags, and description of the transaction, in addition to the identifier.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															

...
...
...
...
guidTX
...
...
...
isoLevel
isoFlags
szDesc
...
...
...
...
...
...
...
...
(szDesc cont'd for 2 rows)

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00006107.
- The **dwcbVarLenData** field MUST be 64.

- guidTX (16 bytes):** This field MUST contain a GUID that specifies the transaction identifier.
- isoLevel (4 bytes):** See the **isoLevel** field in section [2.2.8.1.1.2](#) for details.
- isoFlags (4 bytes):** See the **isoFlags** field in section [2.2.8.1.1.2](#) for details.
- szDesc (40 bytes):** See the **szDesc** field in section [2.2.8.1.1.2](#) for details.

2.2.8.2.2.4.4 TXUSER_IMPORT2_MTAG_SINK_ERROR

This message is sent if the attempt to import the transaction was unsuccessful or to indicate the success or failure of the abort operation.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
Error																															

- MsgHeader (24 bytes):** This field MUST contain a [MESSAGE_PACKET](#) structure.
 - The **dwUserMsgType** field MUST be 0x00006105.
 - The **dwcbVarLenData** field MUST be 4.
- Error (4 bytes):** This field MUST contain the status for the previous request. The value MUST be a member of the [TRUN_TXIMPORT_ERRORS](#) enumeration.

2.2.8.2.2.4.5 TXUSER_IMPORT2_MTAG_SINK_IMPORTED

This message provides the isolation level and isolation flags for the specified transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
isoLevel																															
isoFlags																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00006106.
- The **dwcbVarLenData** field MUST be 8.

isoLevel (4 bytes): See the **isoLevel** field in section [2.2.8.1.1.2](#) for details.

isoFlags (4 bytes): See the **isoFlags** field in section [2.2.8.1.1.2](#) for details.

2.2.8.3 Transaction Administration

2.2.8.3.1 CONNTYPE_TXUSER_GETTXDETAILS

This connection type is used by an application to retrieve details about a transaction from its transaction manager.

For more information about CONNTYPE_TXUSER_GETTXDETAILS as an initiator, see section [3.3.5.3.1](#), and as an acceptor, see section [3.4.5.3.1](#).

2.2.8.3.1.1 TXUSER_GETTXDETAILS_MTAG_GET

This message is used to request details about a transaction from the transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004701.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.8.3.1.2 TXUSER_GETTXDETAILS_MTAG_GOTIT

This message provides the Client with name and identifier details for the transaction superior and all enlisted subordinates.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
ISubordinateCount																															
Reserved																															
vszSuperiorName (variable)																															
...																															
vszSuperiorID (variable)																															
...																															
rgSubordinates (variable)																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004702.
- The **dwcbVarLenData** field MUST be at least 16 bytes.

ISubordinateCount (4 bytes): This field MUST contain the number of subordinates in the **rgSubordinates** array that follows.

Reserved (4 bytes): Reserved. This value MUST be set to zero and MUST be ignored on receipt.

vszSuperiorName (variable): This field MUST contain an [OLETX_VARLEN_STRING](#) structure. The structure specifies an implementation-specific name for the Name property of the superior Enlistment object that is maintained by the [Core Transaction Manager Facet \(section](#)

[1.3.3.3.1](#)) (as specified in [Enlistment Objects \(section 3.2.1.3\)](#)). The Core Transaction Manager Facet is initialized as specified in [Initialization \(section 3.2.3\)](#).

If the transaction manager is the root transaction manager for the transaction, the value MUST be a zero length OLETX_VARLEN_STRING. If the transaction manager is not acting as the root transaction manager for the transaction, the value MUST NOT be a zero length OLETX_VARLEN_STRING. This field MUST be aligned on a 4-byte boundary. Any necessary padding bytes MUST be initialized to an implementation-dependent value, and MUST be ignored on receipt. [<26>](#)

vszSuperiorID (variable): This field contains an OLETX_VARLEN_STRING structure. The structure MUST contain an implementation-specific identifier that corresponds to the identifier property of the superior Enlistment object that is maintained by the Core Transaction Manager Facet (section 1.3.3.3.1) (as specified in Enlistment Objects). The Core Transaction Manager Facet is initialized as specified in Initialization. If the transaction manager is not acting as the root transaction manager for the transaction, the value MUST NOT be a zero length OLETX_VARLEN_STRING. The start of the field MUST be aligned on a 4-byte boundary. Any necessary padding bytes MUST be initialized to an implementation dependent value, and MUST be ignored on receipt. [<27>](#)

rgSubordinates (variable): An array of OLETX_VARLEN_STRING structure pairs. Each pair MUST specify an implementation-specific name, followed by an implementation-specific identifier. [<28>](#) The array MUST contain **ISubordinateCount** pairs of OLETX_VARLEN_STRING structures, representing the collection of subordinates enlisted on the transaction. If **ISubordinateCount** contains zero, this field MUST NOT be present.

The start of each array element MUST be aligned on a 4-byte boundary. Any necessary padding bytes MUST be initialized to an implementation-dependent value, and MUST be ignored on receipt. [<29>](#) The name and identifier correspond to the Name and Identifier properties, respectively, of the Phase One enlistment list that is maintained by the core transaction manager facet, as specified in Enlistment Objects, which is initialized as specified in Initialization.

2.2.8.3.1.3 TXUSER_GETTXDETAILS_MTAG_TX_NOT_FOUND

This message is sent to indicate that the transaction details cannot be found.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004703.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.3.2 CONNTYPE_TXUSER_RESOLVE

This connection type is used by an application either to manually resolve the outcome of an [In Doubt](#) transaction or cause its transaction manager to forget a transaction that is in the [Failed to Notify](#) state.

For more information about CONNTYPE_TXUSER_RESOLVE as an initiator, see section [3.3.5.3.2](#), and as an acceptor, see section [3.4.5.3.2](#).

2.2.8.3.2.1 TXUSER_RESOLVE_MTAG_ACCESSDENIED

This message indicates that the principal that sent the previous [TXUSER_RESOLVE_MTAG_CHILD_ABORT](#), [TXUSER_RESOLVE_MTAG_CHILD_COMMIT](#), or [TXUSER_RESOLVE_MTAG_FORGET_COMMITTED](#) is not authorized to perform the requested action. [<30>](#)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x0000107F.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.3.2.2 TXUSER_RESOLVE_MTAG_CHILD_ABORT

This message is sent by the application to manually resolve the outcome of an in-doubt transaction as aborted.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001071.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.8.3.2.3 TXUSER_RESOLVE_MTAG_CHILD_COMMIT

This message is sent by an application to manually resolve an in-doubt transaction as committed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001072.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.8.3.2.4 TXUSER_RESOLVE_MTAG_CHILD_NOT_PREPARED

This message indicates that the specified transaction is in the [Failed to Notify](#) state, rather than in the [In Doubt](#) state.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001077.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.3.2.5 TXUSER_RESOLVE_MTAG_FORGET_COMMITTED

This message is sent by an application to request that the transaction manager issue a [Forget Transaction](#) event for a transaction that is in the [Failed to Notify](#) state.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001073.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.8.3.2.6 TXUSER_RESOLVE_MTAG_FORGET_TX_NOT_COMMITTED

This message indicates that the specified transaction is in the [In Doubt](#) state, rather than in the [Failed to Notify](#) state.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001078.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.3.2.7 TXUSER_RESOLVE_MTAG_REQUEST_COMPLETE

This message is sent by the transaction manager to indicate that the request completed successfully.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001074.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.3.2.8 TXUSER_RESOLVE_MTAG_TX_NOT_FOUND

This message is sent by the transaction manager to indicate that the specified transaction does not exist.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001075.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.3.3 CONNTYPE_TXUSER_SETTXTIMEOUT

This connection type is used by an application to modify the time-out of a transaction.

For more information about CONNTYPE_TXUSER_SETTXTIMEOUT as an initiator, see section [3.3.5.3.3](#), and as an acceptor, see section [3.4.5.3.3](#).

This connection type also uses the TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE (as specified in section [2.2.8.1.2.6](#)), TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT (as specified in section [2.2.8.1.2.7](#)), and TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE (as specified in section [2.2.8.1.2.8](#)) messages.

2.2.8.3.3.1 TXUSER_SETTXTIMEOUT_MTAG_TX_NOT_FOUND

This message is optionally sent by the transaction manager as part of the [CONNTYPE_TXUSER_SETTXTIMEOUT \(section 2.2.8.3.3\)](#) connection type to indicate that the specified transaction does not exist. This message is optionally also sent by the transaction manager as part of the [CONNTYPE_TXUSER_SETTXTIMEOUT2 \(section 2.2.8.3.4\)](#) connection type to indicate that the transaction manager supports the capability to modify the time-out of a transaction. For more information, see CONNTYPE_TXUSER_SETTXTIMEOUT2.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x0000107D.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.3.4 CONNTYPE_TXUSER_SETTXTIMEOUT2

This connection type is used by an application to modify the time-out of a transaction. This connection type supersedes [CONNTYPE_TXUSER_SETTXTIMEOUT](#).

For more information about CONNTYPE_TXUSER_SETTXTIMEOUT2 as an initiator, see section [3.3.5.3.4](#), and as an acceptor, see section [3.4.5.3.4](#).

This connection type also uses the TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT (see section [2.2.8.1.2.7](#)), TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE (see section [2.2.8.1.2.8](#)), and TXUSER_SETTXTIMEOUT_MTAG_TX_NOT_FOUND (see section [2.2.8.3.3.1](#)) messages.

2.2.8.3.5 CONNTYPE_TXUSER_TRACE

This connection type is used by an application to ask its transaction manager to trace the status of a transaction by using an implementation-specific mechanism. [<31>](#)

For more information about CONNTYPE_TXUSER_TRACE as an initiator, see section [3.3.5.3.5](#), and as an acceptor, see section [3.4.5.3.5](#).

2.2.8.3.5.1 TXUSER_TRACE_MTAG_DUMP_TRANSACTION

This message requests the transaction manager write the status of a transaction to a local trace file in an implementation-specific manner.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002100.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.8.3.5.2 TXUSER_TRACE_MTAG_REQUEST_COMPLETE

This message indicates the transaction was successfully traced.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002101.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.3.5.3 TXUSER_TRACE_MTAG_REQUEST_FAILED

This message indicates that the trace request failed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002103.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.3.5.4 TXUSER_TRACE_MTAG_TX_NOT_FOUND

This message is sent by the transaction manager to indicate that the trace request failed because the transaction does not exist.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002102.
- The **dwcbVarLenData** field MUST be 0.

2.2.8.4 Transaction Manager Administration

2.2.8.4.1 CONNTYPE_TXUSER_GETSECURITYFLAGS

This connection type is used by an application to obtain the security configuration of its transaction manager.

For more information about CONNTYPE_TXUSER_GETSECURITYFLAGS as an initiator, see section [3.3.5.4.1](#), and as an acceptor, see section [3.4.5.3.5](#).

2.2.8.4.1.1 TXUSER_GETSECURITYFLAGS_MTAG_FETCHED

This message indicates that the request to obtain security configuration flags was successful.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
grfNetworkDtcAccess																															
grfXaTransactions																															
grfReserved3																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00005502.
- The **dwcbVarLenData** field MUST be 12.

grfNetworkDtcAccess (4 bytes): This field contains a [DTCADVCONFIG](#) bitfield enumeration, see DTCADVCONFIG for details.

grfXaTransactions (4 bytes): This field indicates whether the transaction manager supports the local use of the XA standard API in an implementation-specific manner as specified in [Receiving a TXUSER_GETSECURITYFLAGS_MTAG_GETSECURITYFLAGS Message \(section 3.4.5.4.1.1\)](#). For more information about XA, see [\[C193\]](#). The field SHOULD have a value of zero if the use of the XA standard API is not supported; or it SHOULD have a value of one if the use of the XA standard API is supported.

grfReserved3 (4 bytes): Reserved. This value MUST be set to zero, and MUST be ignored on receipt.

2.2.8.4.1.2 TXUSER_GETSECURITYFLAGS_MTAG_GETSECURITYFLAGS

This message is used by an application to obtain the configuration flags that are associated with the security settings of a transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00005501.
- The **dwcbVarLenData** field MUST be 0.

2.2.9 Connection Types Relevant to Transaction Managers

2.2.9.1 Transaction Propagation and Coordination

2.2.9.1.1 Push Propagation

2.2.9.1.1.1 CONNTYPE_PARTNERTM_PROPAGATE

This connection type is used by a superior transaction manager to do a [Push Propagation](#) of a transaction to its subordinate transaction manager and to execute the Two-Phase Commit protocol.

For more information about [CONNTYPE_PARTNERTM_PROPAGATE as Initiator](#), see section [3.7.5.1.1.1](#), and as an acceptor, see section [CONNTYPE_PARTNERTM_PROPAGATE as Acceptor \(section 3.8.5.1.1.1\)](#).

2.2.9.1.1.1.1 PARTNERTM_PROPAGATE_MTAG_PROPAGATE

This message is used to propagate a transaction to a subordinate transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															

...
...
...
...
guidTX
...
...
...
isoLevel
szDesc
...
...
...
...
...
...
...
...
(szDesc cont'd for 2 rows)

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002001.
- The **dwcbVarLenData** field MUST be 60.

guidTX (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

isoLevel (4 bytes): See the **isoLevel** field in section [2.2.8.1.1.2](#) for details.

szDesc (40 bytes): See the **szDesc** field in section [2.2.8.1.1.2](#) for details.

2.2.9.1.1.1.2 PARTNERTM_PROPAGATE_MTAG_PROPAGATED

This message indicates that the transaction was successfully propagated to the subordinate transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002002.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.3 PARTNERTM_PROPAGATE_MTAG_DUPLICATE

This message indicates that the transaction was already propagated to the subordinate transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002010.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.4 PARTNERTM_PROPAGATE_MTAG_NO_MEM

This message indicates that transaction propagation failed because the subordinate transaction manager was out of memory.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002901.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.5 PARTNERTM_PROPAGATE_MTAG_LOG_FULL

This message indicates that transaction propagation failed because the transaction recovery log of the subordinate transaction manager is full.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

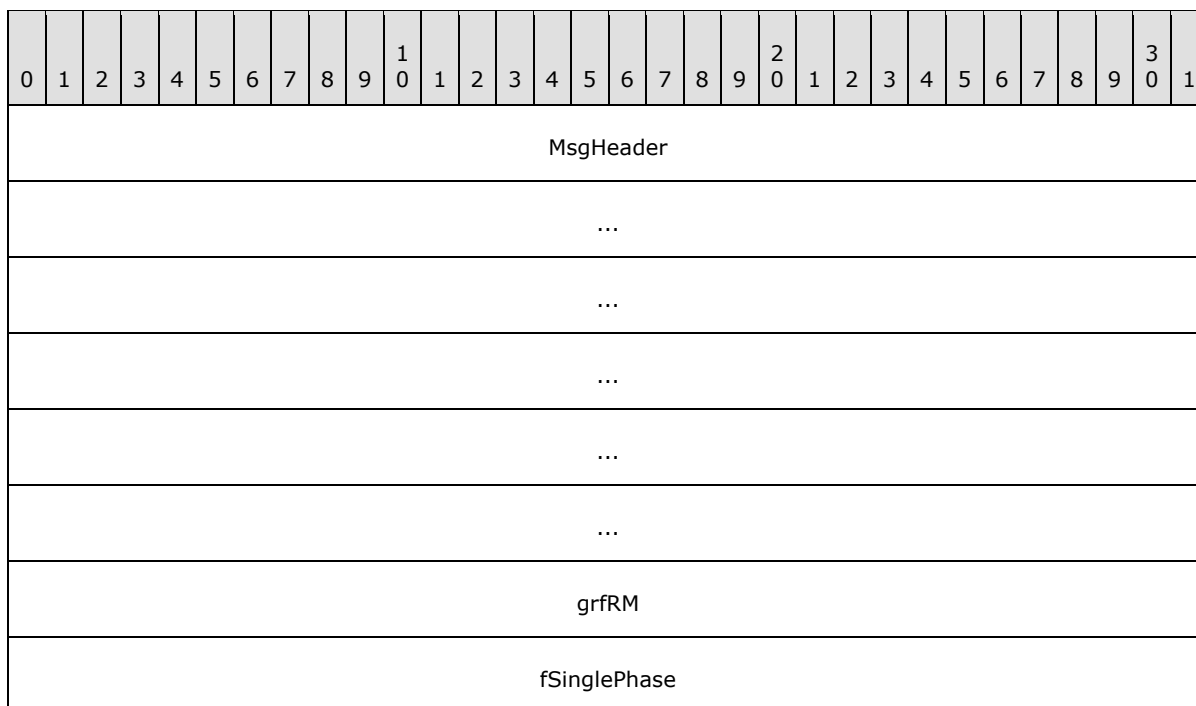
MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002902.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.6 PARTNERTM_PROPAGATE_MTAG_PREPAREREQ

This message is used to request that the subordinate transaction manager perform the actions that are needed to prepare the transaction to be committed.

This message is also used for [CONNTYPE PARTNERTM_BRANCH](#).



MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002003.
- The **dwcbVarLenData** field MUST be 8.

grfRM (4 bytes): The value of this field MUST be as specified in [GRFRM \(section 2.2.7.1\)](#).

fSinglePhase (4 bytes): Indicates whether the sending transaction manager will allow the single-phase commit optimization. If the value is zero, the receiver of the message MUST NOT perform a single-phase commit for its superior transaction manager. If the value is nonzero, the receiver SHOULD perform a single-phase commit for its superior transaction manager.

2.2.9.1.1.1.7 PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE

This message indicates that the subordinate transaction manager has processed the Prepare request from the superior transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
prepareReqDone																															
guidReason																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002006.
- The **dwcbVarLenData** field MUST be 20.

prepareReqDone (4 bytes): Indicates the status of the Prepare request as specified in the [PARTNERTM_PROPAGATE_PREPAREREQDONE_RESPONSE \(section 2.2.6.4\)](#) enumeration.

guidReason (16 bytes): Reserved. This value SHOULD be set to a NULL GUID and MUST be ignored on receipt.

2.2.9.1.1.1.8 PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR

This message indicates that the sender detected a violation of the Two-Phase Commit protocol and is unable to perform the previous request it received over the connection.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002009.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.9 PARTNERTM_PROPAGATE_MTAG_COMMITREQ

This message is sent by the superior transaction manager to request that the transaction be committed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002005.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.10 PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE

This message indicates the transaction was successfully committed by the subordinate transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002008.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.11 PARTNERTM_PROPAGATE_MTAG_ABORTREQ

This message is sent by the superior transaction manager to request that the transaction be aborted.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002004.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.12 PARTNERTM_PROPAGATE_MTAG_ABORTREQDONE

This message is sent by the subordinate transaction manager to indicate that the transaction was successfully aborted.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002007.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.13 PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY

This message is sent to abort a transaction before the [PARTNERTM_PROPAGATE_MTAG_PREPAREREQ \(section 2.2.9.1.1.1.6\)](#) message is received.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002903.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.14 PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER

This message is sent by the subordinate transaction manager to register for a [Phase Zero](#) notification.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002906.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.15 PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTERED

This message is sent by the superior transaction manager to indicate that the subordinate transaction manager was successfully registered for [Phase Zero](#) notifications.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002907.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.16 PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTRATIONREJECTED

This message is sent by the superior transaction manager to indicate that it was unable to register the subordinate transaction manager for [Phase Zero](#) notifications.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002910.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.17 PARTNERTM_PROPAGATE_MTAG_PHASE0

This message is sent by the superior transaction manager to request that the subordinate transaction manager begin [Phase Zero](#).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002908.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.1.1.18 PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE

This message indicates that the subordinate transaction manager successfully completed [Phase Zero](#).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002909.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.2 Pull Propagation

2.2.9.1.2.1 CONNTYPE_PARTNERTM_BRANCH

A subordinate transaction manager uses this connection type to register a new subordinate enlistment with a superior transaction manager.

The two transaction managers also use this connection type to execute the Two-Phase Commit protocol. This connection type is initiated as a result of a [TXUSER_ASSOCIATE MTAG_ASSOCIATE](#) message that is sent by an application to the subordinate transaction manager to request a Pull Propagation operation.

For more information about CONNTYPE_PARTNERTM_BRANCH as an initiator, see section [3.8.5.1.2.1](#), and as an acceptor, see section [3.7.5.1.2.1](#).

This connection type also uses [PARTNERTM_PROPAGATE MTAG_PREPAREREQ](#), [PARTNERTM_PROPAGATE MTAG_PREPAREREQDONE](#), [PARTNERTM_PROPAGATE MTAG_COMMITREQ](#), [PARTNERTM_PROPAGATE MTAG_COMMITREQDONE](#), [PARTNERTM_PROPAGATE MTAG_ABORTREQ](#), [PARTNERTM_PROPAGATE MTAG_ABORTREQDONE](#), [PARTNERTM_PROPAGATE MTAG_ABORTNOTIFY](#), [PARTNERTM_PROPAGATE MTAG_PHASE0REGISTER](#), [PARTNERTM_PROPAGATE MTAG_PHASE0REGISTERED](#), [PARTNERTM_PROPAGATE MTAG_PHASE0REGISTRATIONREJECTED](#), [PARTNERTM_PROPAGATE MTAG_PHASE0](#), and [PARTNERTM_PROPAGATE MTAG_PHASE0COMPLETE](#) messages.

2.2.9.1.2.1.1 PARTNERTM_BRANCH_MTAG_BRANCH_LOG_FULL

This message indicates that the branch request failed because the transaction recovery log of the superior transaction manager is full.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002056.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.2.1.2 PARTNERTM_BRANCH_MTAG_BRANCH_NO_MEM

This message indicates that the branch request failed because the superior transaction manager was unable to allocate sufficient memory.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002057.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.2.1.3 PARTNERTM_BRANCH_MTAG_BRANCH_TOO_LATE

This message indicates that the branch request failed because it was too late in the transaction life cycle. For more information, see [Create Subordinate Enlistment \(section 3.2.7.11\)](#) and [Create Subordinate Enlistment Failure \(section 3.7.7.7\)](#).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002055.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.2.1.4 PARTNERTM_BRANCH_MTAG_BRANCH_TOO_MANY

This message indicates that the branch request failed because the superior transaction manager has reached the maximum number of subordinates allowed on a transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002059.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.2.1.5 PARTNERTM_BRANCH_MTAG_BRANCH_TX_NOT_FOUND

This message indicates that the branch request failed because the superior transaction manager was unaware of the transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002054.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.2.1.6 PARTNERTM_BRANCH_MTAG_BRANCHED

This message is sent by the superior transaction manager to indicate that the branch request was successful.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002052.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.1.2.1.7 PARTNERTM_BRANCH_MTAG_BRANCHING

This message is sent by a subordinate transaction manager to register a new subordinate enlistment with a superior transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTX																															
...																															
...																															
...																															

- MsgHeader (24 bytes):** This field MUST contain a [MESSAGE_PACKET](#) structure.
- The **dwUserMsgType** field MUST be 0x00002051.
 - The **dwcbVarLenData** field MUST be 16.
- guidTX (16 bytes):** This field MUST contain a GUID that specifies the transaction identifier.

2.2.9.2 Transaction Recovery

2.2.9.2.1 Subordinate-Driven

2.2.9.2.1.1 CONNTYPE_PARTNERTM_CHECKABORT

This connection type is used by a subordinate transaction manager to query the outcome of a transaction from its superior transaction manager.

For more information about CONNTYPE_PARTNERTM_CHECKABORT as an initiator, see section [3.8.5.2.1.1](#), and as an acceptor, see section [3.7.5.2.1.1](#).

2.2.9.2.1.1.1 PARTNERTM_CHECKABORT_MTAG_CHECK

This message is used by a subordinate transaction manager to check if the superior transaction manager aborted a specific transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTX																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002021.
- The **dwcbVarLenData** field MUST be 16.

guidTX (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.9.2.1.1.2 PARTNERTM_CHECKABORT_MTAG_ABORTED

This message indicates that the transaction was successfully aborted.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002022.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.2.1.1.3 PARTNERTM_CHECKABORT_MTAG_RETRY

This message indicates the superior transaction manager is unable to declare that the transaction aborted, either because the superior transaction manager has not yet determined the final outcome of the transaction, or because the transaction has already committed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002023.

- The **dwcbVarLenData** field MUST be 0.

2.2.9.2.2 Superior-Driven

2.2.9.2.2.1 CONNTYPE_PARTNERTM_REDELIVERCOMMIT

This connection type is used by a superior transaction manager to redeliver a Commit notification for a transaction to its subordinate transaction manager.

For more information about CONNTYPE_PARTNERTM_REDELIVERCOMMIT as an initiator, see section [3.7.5.2.2.1](#), and as an acceptor, see section [3.8.5.2.2.1](#).

2.2.9.2.2.1.1 PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQ

The superior transaction manager sends this message to begin [Phase Two](#) commit processing.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002011.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.9.2.2.1.2 PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQDONE

This message indicates that the subordinate transaction manager has successfully committed the transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002012.
- The **dwcbVarLenData** field MUST be 0.

2.2.9.2.2.1.3 PARTNERTM_REDELIVERCOMMIT_MTAG_RETRY

This message is sent by the subordinate transaction manager to indicate that it is in a state in which it is temporarily unable to process the commit request.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002013.
- The **dwcbVarLenData** field MUST be 0.

2.2.10 Connection Types Relevant to Resource Managers

2.2.10.1 Resource Manager Registration

2.2.10.1.1 CONNTYPE_TXUSER_RESOURCEMANAGER

The CONNTYPE_TXUSER_RESOURCEMANAGER connection type is used by a durable resource manager to register with its transaction manager.

For more details about CONNTYPE_TXUSER_RESOURCEMANAGER as an initiator, see section [3.5.5.1.1](#), and as an acceptor, see section [3.6.5.1.1](#).

2.2.10.1.1.1 TXUSER_RESOURCEMANAGER_MTAG_CREATE

The TXUSER_RESOURCEMANAGER_MTAG_CREATE message is used by resource managers to register with a transaction manager.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidRM																															
...																															
...																															
...																															
guidSession																															
...																															
...																															
...																															

- MsgHeader (24 bytes):** This field MUST contain a [MESSAGE_PACKET](#) structure.
- The **dwUserMsgType** field MUST be 0x00001051.
 - The **dwcbVarLenData** field MUST be 32.
- guidRM (16 bytes):** This field MUST contain a GUID that specifies the **resource manager identifier**.
- guidSession (16 bytes):** This field MUST contain a GUID that specifies the session identifier of the resource manager.

2.2.10.1.1.2 TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE

The TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE message is sent from the transaction manager when there is already a resource manager that is registered with the same **guidRM** value.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001054.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.1.1.3 TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE

The TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE message is used by resource managers to inform the transaction manager that it has received outcome from the transaction manager for all outstanding in-doubt transactions for which the resource manager required an outcome.

This message is used in the following scenarios:

- [Recover Transactions \(section 3.5.7.2\)](#)
- [Recover Transaction \(section 3.5.7.1\)](#)
- [Reenlistment Complete \(section 3.5.7.3\)](#)
- [Enlistment Commit Request Completed \(section 3.5.4.5\)](#)
- [Enlistment Abort Request Completed \(section 3.5.4.4\)](#)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001052.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.1.1.4 TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE

The TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE message is used by transaction managers to indicate that the previous request that was sent by the resource manager on the connection was successfully completed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001053.

- The **dwcbVarLenData** field MUST be 0.

2.2.10.1.2 CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL

This connection type is used by a durable resource manager to register with a transaction manager as well as to detect duplicate registrations. This connection type supersedes [CONNTYPE_TXUSER_RESOURCEMANAGER](#).

For more information about CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL as an initiator, see section [3.5.5.1.2](#), and as an acceptor, see section [3.6.5.1.2](#).

2.2.10.1.2.1

TXUSER_RESOURCEMANAGERINTERNAL_MTAG_DUPLICATEDETECTED

This message notifies a resource manager that an attempt was made to register another resource manager instance with the unique identifier of this resource manager. See the **guidRM** field in [TXUSER_RESOURCEMANAGER_MTAG_CREATE](#) for more information.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001055.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2 Transaction Coordination

2.2.10.2.1 CONNTYPE_TXUSER_PHASE0

This CONNTYPE_TXUSER_PHASE0 connection type is used by a resource manager to enlist for [Phase Zero](#) notifications from its transaction manager.

The CONNTYPE_TXUSER_PHASE0 connection type may be used as either an initiator or as an acceptor:

- For more details about CONNTYPE_TXUSER_PHASE0 as an initiator, see section [3.5.5.2.1](#).

- For more details about CONNTYPE_TXUSER_PHASE0 as an acceptor, see section [3.6.5.1.1](#).

2.2.10.2.1.1 TXUSER_PHASE0_MTAG_CREATE

The TXUSER_PHASE0_MTAG_CREATE message is sent by a resource manager to a transaction manager to create a new Phase Zero enlistment on a transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004901.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.10.2.1.2 TXUSER_PHASE0_MTAG_CREATE_TOO_LATE

The TXUSER_PHASE0_MTAG_CREATE_TOO_LATE message is sent by the transaction manager if the creation of the Phase Zero enlistment failed because the enlistment request was made too late in the specified **transaction lifetime**. See [Create Phase Zero Enlistment Failure \(section 3.6.7.7\)](#) and [Register Phase Zero Failure \(section 3.2.7.28\)](#) for more information.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004907.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.1.3 TXUSER_PHASE0_MTAG_CREATE_TX_NOT_FOUND

The TXUSER_PHASE0_MTAG_CREATE_TX_NOT_FOUND message is sent by the transaction manager if the creation of the Phase Zero enlistment failed because the specified transaction could not be found.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004906.

- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.1.4 TXUSER_PHASE0_MTAG_CREATED

The TXUSER_PHASE0_MTAG_CREATED message is sent by the transaction manager if the creation of the Phase Zero enlistment was successful.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004902.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.1.5 TXUSER_PHASE0_MTAG_PHASE0REQ

The TXUSER_PHASE0_MTAG_PHASE0REQ message indicates a [Phase Zero](#) request from the transaction manager to the Phase Zero enlistment.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004903.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.1.6 TXUSER_PHASE0_MTAG_PHASE0REQ_ABORT

The TXUSER_PHASE0_MTAG_PHASE0REQ_ABORT message is sent by the transaction manager to notify the Phase Zero enlistment that the transaction aborted.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004909.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.1.7 TXUSER_PHASE0_MTAG_PHASE0REQDONE

The TXUSER_PHASE0_MTAG_PHASE0REQDONE message is sent by the resource manager to notify the transaction manager that the Phase Zero enlistment has completed the [Phase Zero](#) processing request.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004904.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.1.8 TXUSER_PHASE0_MTAG_UNENLIST

The TXUSER_PHASE0_MTAG_UNENLIST message is sent by the resource manager to notify the transaction manager that the Phase Zero enlistment is to be removed and is no longer part of the transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00004905.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.2 CONNTYPE_TXUSER_ENLISTMENT

The CONNTYPE_TXUSER_ENLISTMENT connection type is used by a durable resource manager to establish an enlistment with its transaction manager.

For more details about CONNTYPE_TXUSER_ENLISTMENT as an initiator, see section [3.5.5.2.2](#), and as an acceptor, see section [3.6.5.2.2](#).

2.2.10.2.2.1 TXUSER_ENLISTMENT_MTAG_ABORTREQ

The TXUSER_ENLISTMENT_MTAG_ABORTREQ message is sent by the transaction manager (TM) to the resource manager (RM) to inform the RM that the transaction has aborted.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001034.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.2.2 TXUSER_ENLISTMENT_MTAG_ABORTREQDONE

The TXUSER_ENLISTMENT_MTAG_ABORTREQDONE message acknowledges that the resource manager processed the abort and the transaction manager is no longer obligated to retain the outcome of the transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001037.

- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.2.3 TXUSER_ENLISTMENT_MTAG_COMMITREQ

The TXUSER_ENLISTMENT_MTAG_COMMITREQ message is sent by the transaction manager to notify the resource manager that the transaction has committed and that the resource manager MUST carry out the operations that are necessary to commit the work that is performed under the transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001035.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.2.4 TXUSER_ENLISTMENT_MTAG_COMMITREQDONE

The TXUSER_ENLISTMENT_MTAG_COMMITREQDONE message is sent by the resource manager to indicate that it has carried out the necessary operations to commit the transaction, and that the transaction manager is no longer obligated to retain the outcome of the transaction for the resource manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001038.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.2.5 TXUSER_ENLISTMENT_MTAG_ENLIST

The TXUSER_ENLISTMENT_MTAG_ENLIST message is sent by the resource manager to request the creation of a new enlistment on a transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTX																															
...																															

...
...
guidRM
...
...
...
guidSession
...
...
...

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001031.
- The **dwcbVarLenData** field MUST be 48.

guidTX (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

guidRM (16 bytes): This field MUST contain a GUID that specifies the resource manager identifier. See the **guidRM** field in the [TXUSER_RESOURCEMANAGER_MTAG_CREATE \(section 2.2.10.1.1.1\)](#) message for details.

guidSession (16 bytes): This field MUST contain a GUID that specifies the session identifier of the resource manager. See the **guidSession** field in the [TXUSER_RESOURCEMANAGER_MTAG_CREATE \(section 2.2.10.1.1.1\)](#) for details.

2.2.10.2.2.6 TXUSER_ENLISTMENT_MTAG_ENLIST_LOG_FULL

The TXUSER_ENLISTMENT_MTAG_ENLIST_LOG_FULL message is sent by the transaction manager to indicate that the creation of the new enlistment failed because insufficient space exists in the recovery log of the transaction manager to be able to account for the new enlistment.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001903.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.2.7 TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_LATE

The TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_LATE message is sent by the transaction manager to indicate that the creation of that enlistment failed because it is too late in the lifetime of the specified transaction. See [Create Subordinate Enlistment Failure](#) and [Create Subordinate Enlistment](#) for more information.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001902.

- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.2.8 TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_MANY

The TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_MANY message is sent by the transaction manager to indicate that the creation of the new enlistment failed because the implementation-specific maximum number of enlistments for the transaction has been reached. [<32>](#)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001905.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.2.9 TXUSER_ENLISTMENT_MTAG_ENLIST_TX_NOT_FOUND

The TXUSER_ENLISTMENT_MTAG_ENLIST_TX_NOT_FOUND message is sent by the transaction manager to indicate that the creation of the new enlistment failed because the specified transaction does not exist.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001901.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.2.10 TXUSER_ENLISTMENT_MTAG_ENLISTED

The TXUSER_ENLISTMENT_MTAG_ENLISTED message is sent by the transaction manager to indicate that the creation of the new enlistment was successful.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001032.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.2.2.11 TXUSER_ENLISTMENT_MTAG_PREPAREREQ

The TXUSER_ENLISTMENT_MTAG_PREPAREREQ message is used to request that the resource manager perform the actions that are needed to prepare the transaction to be committed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
grfRM																															
fSinglePhase																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001033.
- The **dwcbVarLenData** field MUST be 8.

grfRM (4 bytes): The value of this field MUST be as specified in [GRFRM \(section 2.2.7.1\)](#).

fSinglePhase (4 bytes): Indicates whether the sending transaction manager is willing to allow the single-phase commit optimization. If the value is zero, the resource manager receiving this message MUST NOT perform a single-phase commit. If the value is non-zero, the resource manager receiving this message SHOULD perform a single-phase commit.

2.2.10.2.2.12 TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE

The TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE message is sent by the resource manager to indicate either success or failure of the prepare operation, depending on the value of the **prepareReqDone** field.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
prepareReqDone																															
guidReason																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001036.
- The **dwcbVarLenData** field MUST be 20.

prepareReqDone (4 bytes): A value indicating the result of the prepare operations that are performed by the resource manager. The value MUST be one that is as specified by the [TXUSER_ENLISTMENT_PREPAREREQDONE_RESPONSE](#) enumeration.

guidReason (16 bytes): This field MUST contain a GUID that contains an implementation-specific value that MUST be ignored on receipt. [<33>](#)

2.2.10.3 Transaction Recovery

2.2.10.3.1 CONNTYPE_TXUSER_REENLIST

This connection type is used by a durable resource manager to determine the outcome of an [In Doubt](#) transaction.

For more information about CONNTYPE_TXUSER_REENLIST as an initiator, see section [3.5.5.3.1](#), and as an acceptor, see section [3.6.5.3.1](#).

2.2.10.3.1.1 TXUSER_REENLIST_MTAG_REENLIST

The TXUSER_REENLIST_MTAG_REENLIST message indicates that the resource manager wants to obtain the outcome of an [In Doubt](#) transaction from the transaction manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															
ulTimeout																															
guidRm																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001061.
- The **dwcbVarLenData** field MUST be 36.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

ulTimeout (4 bytes): This field MUST specify the time, in milliseconds, that the resource manager will wait for a decision. A value of zero MUST represent an infinite timeout. The recipient SHOULD NOT send a [TXUSER_REENLIST_MTAG_REENLIST_TIMEOUT](#) message until the time-span that is specified by this value has elapsed.

guidRm (16 bytes): This field MUST be a GUID that specifies the resource manager identifier. See the **guidRM** field in the [TXUSER_RESOURCEMANAGER_MTAG_CREATE](#) message for more information.

2.2.10.3.1.2 TXUSER_REENLIST_MTAG_REENLIST_ABORTED

The TXUSER_REENLIST_MTAG_REENLIST_ABORTED message indicates that the transaction that is supplied by the [TXUSER_REENLIST_MTAG_REENLIST](#) has aborted.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001062.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.3.1.3 TXUSER_REENLIST_MTAG_REENLIST_COMMITTED

The TXUSER_REENLIST_MTAG_COMMITTED message indicates that the transaction that is supplied by the [TXUSER_REENLIST_MTAG_REENLIST](#) has committed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001063.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.3.1.4 TXUSER_REENLIST_MTAG_REENLIST_TIMEOUT

The TXUSER_REENLIST_MTAG_REENLIST_TIMEOUT message indicates that the [TXUSER_REENLIST_MTAG_REENLIST](#) request has exceeded the time-span that is specified by its **ulTimeout** field and therefore, has failed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001064.

- The **dwcbVarLenData** field MUST be 0.

2.2.10.4 Voting

2.2.10.4.1 CONNTYPE_TXUSER_VOTER

This connection type is used by a Volatile resource manager to establish a Voter Enlistment with its transaction manager.

For more details on CONNTYPE_TXUSER_VOTER as an initiator, see section [3.5.5.4.1](#), and as an acceptor, see section [3.6.5.4.1](#).

2.2.10.4.1.1 TXUSER_STATUS_MTAG_ABORTED

The TXUSER_STATUS_MTAG_ABORTED message is sent by the transaction manager to notify the resource manager that the transaction has aborted.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001093.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.4.1.2 TXUSER_STATUS_MTAG_COMMITTED

The TXUSER_STATUS_MTAG_COMMITTED message is sent by the transaction manager to notify the resource manager that the transaction has committed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001094.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.4.1.3 TXUSER_STATUS_MTAG_INDOUBT

The TXUSER_STATUS_MTAG_INDOUBT message is sent by the transaction manager to notify the resource manager that the outcome of the transaction is [In Doubt](#).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00001095.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.4.1.4 TXUSER_VOTER_MTAG_CREATE

The TXUSER_VOTER_MTAG_CREATE message is sent by the transaction manager to create a new voter enlistment on a transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
guidTx																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002091.
- The **dwcbVarLenData** field MUST be 16.

guidTx (16 bytes): This field MUST contain a GUID that specifies the transaction identifier.

2.2.10.4.1.5 TXUSER_VOTER_MTAG_CREATE_TOO_LATE

The TXUSER_VOTER_MTAG_CREATE_TOO_LATE message is sent by the transaction manager to indicate that the creation of the new voter enlistment was unsuccessful because it was too late in the lifetime of the transaction to create new enlistments. See [Create Voter Enlistment Failure \(section 3.6.7.12\)](#) and [Create Voter Enlistment \(section 3.2.7.14\)](#) for more information.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002096.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.4.1.6 TXUSER_VOTER_MTAG_CREATE_TX_NOT_FOUND

The TXUSER_VOTER_MTAG_CREATE_TX_NOT_FOUND message is sent by the transaction manager to indicate that creation of the new voter enlistment was unsuccessful because the specified transaction does not exist.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002095.

- The **dwcbVarLenData** field MUST be 0.

2.2.10.4.1.7 TXUSER_VOTER_MTAG_CREATED

The TXUSER_VOTER_MTAG_CREATED message is sent by the transaction manager to indicate that creation of the new voter enlistment was successful.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002092.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.4.1.8 TXUSER_VOTER_MTAG_VOTEREQ

The TXUSER_VOTER_MTAG_VOTEREQ message is sent by the transaction manager to request that the resource manager performs any operations it needs to during [Phase One](#) and to vote on the outcome of the transaction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002093.
- The **dwcbVarLenData** field MUST be 0.

2.2.10.4.1.9 TXUSER_VOTER_MTAG_VOTEREQDONE

The TXUSER_VOTER_MTAG_VOTEREQDONE message is sent by a voter to indicate whether it agrees to a decision to commit the transaction for which it had previously created a voter enlistment.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MsgHeader																															
...																															
...																															
...																															
...																															
...																															
VoteReqDone																															

MsgHeader (24 bytes): This field MUST contain a [MESSAGE_PACKET](#) structure.

- The **dwUserMsgType** field MUST be 0x00002094.
- The **dwcbVarLenData** field MUST be 4.

VoteReqDone (4 bytes): The resource manager vote to commit or abort the transaction. The value MUST be one that is defined by the [TXUSER_VOTER_VOTERREQDONE_RESPONSE](#) enumeration.

3 Protocol Details

3.1 Common Details

This section defines common details for the transaction participants, as specified in sections [3.2](#) through [3.8](#). Each participant MUST conform to the details as specified in this section.

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with the behavior that is described in this document.

Note that the abstract data model can be implemented in a variety of ways. This protocol does not prescribe or advocate any specific implementation technique.

A participant MUST maintain all the data elements, as specified in [\[MS-CMP\]](#) section 3.1.1.

Participants MUST use the MSDTC Connection Manager: OleTx Multiplexing Protocol connections (as specified in [\[MS-CMP\]](#) section 3.1.1.1) as a transport protocol for sending messages. The [Transport](#) section defines the mechanisms by which this protocol initializes and makes use of the MSDTC Connection Manager: OleTx Multiplexing Protocol .

A participant MUST also maintain the following data elements:

- Transaction table: A table of entries to transaction objects, keyed by transaction identifier.

Each transaction object MUST contain the following data structures:

- Identifier: This field contains a GUID that specifies the transaction identifier.
- Connection list: A list of MSDTC Connection Manager: OleTx Multiplexing Protocol connection objects (as specified in [\[MS-CMP\]](#) section 3.1.1.1) that are associated with the transaction.

Furthermore, a participant MUST extend the definition of a connection object to include the following data elements:

- Transaction: A reference to the transaction object that is associated with the connection.
- State: A state enumeration that represents the current state of the connection.

A state enumeration MUST contain a set of values that represent specific states in a logical state machine. For a connection type, these values represent the different states to which the connection's logical state machine is set during the lifetime of the connection.

When a participant initiates or accepts a connection, the State field of the connection MUST be set initially to the Idle state. When the connection is disconnected, the connection state MUST be set to the Ended state.

For a participant initiating a connection, once the connection's state machine enters the Ended state, the connection that is associated with the state machine MUST be disconnected, if it is not already disconnected, as specified in [\[MS-CMP\]](#) section 3.1.5.1.

A participant MUST support both initiating and accepting multiple concurrent connections of any type inside the same MSDTC Connection Manager: OleTx Transports Protocol session (as specified in

[\[MS-CMPO\]](#) section 3.2.1.2) or different MSDTC Connection Manager: OleTx Transports Protocol sessions. Consequently, a participant MUST support the existence of multiple instances of a single connection of the same type. A participant MUST also support initiating multiple concurrent sessions to a number of different endpoints.

3.1.1.1 Converting a Name Object to an OLETX_TM_ADDR Structure

A Name object MUST be converted to an [OLETX_TM_ADDR \(section 2.2.4.2\)](#) structure in the following manner:

- The **guidSignature** field of OLETX_TM_ADDR MUST be set as specified in section [2.2.4.2](#).
- The **guidEndpoint** field of OLETX_TM_ADDR MUST be set to the Endpoint field of the Name object.
- The **grbComProtsSupported** field of OLETX_TM_ADDR MUST be set to the Protocols field of the Name object.
- The **wszHostName** field of OLETX_TM_ADDR MUST be set to the Hostname field of the Name object.

3.1.1.2 Converting an OLETX_TM_ADDR Structure to a Name Object

An [OLETX_TM_ADDR \(section 2.2.4.2\)](#) structure MUST be converted to a Name object in the following manner:

- The **Endpoint** field of the Name object MUST be set to the **guidEndpoint** field of OLETX_TM_ADDR.
- The **Protocols** field of the Name object MUST be set to the **grbComProtsSupported** field of OLETX_TM_ADDR.
- The **Hostname** field of the Name object MUST be set to the **wszHostName** field of OLETX_TM_ADDR.

3.1.2 Timers

None.

3.1.3 Initialization

To establish an OleTx connection between an initiator and an acceptor, both the initiator and the acceptor MUST follow the initialization steps specified in [\[MS-CMPO\]](#) section 1.3.3.1.

To initiate a connection, a session MUST already be established between the initiator and the acceptor.

For the use of MSDTC Connection Manager: OleTx Transports Protocol sessions (as specified in [\[MS-CMPO\]](#) section 3.2.1.2) and MSDTC Connection Manager: OleTx Multiplexing Protocol connections (as specified in [\[MS-CMP\]](#) section 3.1.1.1) in this protocol, see section [2.1](#).

3.1.4 Protocol Versioning Details

3.1.4.1 Supporting a Protocol Version

A protocol role implementation that claims support for a protocol version MUST implement all the protocol elements required by that version for the respective role, as specified in section [2.2.1](#).

A protocol role implementation that claims a version as the maximum supported protocol version MUST support that version, and it MUST NOT implement any protocol elements that are neither required nor optional for that version (see section [2.2.1](#)).

3.1.4.2 Negotiating a Common Protocol Version

Before exchanging any protocol messages, two protocol participants MUST agree on what protocol version to use for their communication. To negotiate a common protocol version, the two protocol participants MUST use the version negotiation mechanism provided by the MSDTC Connection Manager: OleTx Transports Protocol transport (see [\[MS-CMPO\]](#) section **3.1.4.2.1** BuildContext-Primary) as follows:

- When a protocol participant (Application, Resource Manager, Transaction Manager) initializes its underlying MSDTC Connection Manager: OleTx Transports Protocol transport, it MUST do the following:
 - Set the Minimum Level 3 Version Number data field of the underlying MSDTC Connection Manager: OleTx Transports Protocol implementation to 0x00000001 (see also [\[MS-CMPO\]](#), section [3.2.1.1](#)). Note that the MSDTC Connection Manager: OleTx Transaction Protocol is layered on top of MSDTC Connection Manager: OleTx Multiplexing Protocol (specified in [\[MS-CMP\]](#)), which is layered on top of the MSDTC Connection Manager: OleTx Transports Protocol (specified in [\[MS-CMPO\]](#)). Therefore, it is a level-three protocol for the MSDTC Connection Manager: OleTx Transports Protocol (as defined in [\[MS-CMPO\]](#), section [2.2.4](#)).
 - Set the Maximum Level 3 Version Number data field of the underlying MSDTC Connection Manager: OleTx Transports Protocol implementation to the value of the maximum supported MSDTC Connection Manager: OleTx Transaction Protocol version (defined in section [3.1.4.1](#)).

When an MSDTC Connection Manager: OleTx Transports Protocol session is successfully established between the two protocol participants, the value of the **dwLevelThreeAccepted** field of the session object's **Version** field (see [\[MS-CMPO\]](#) section 3.1.1.2 Session State) indicates the negotiated protocol version (for example, if the value of the **dwLevelThreeAccepted** field is 5, the negotiated protocol version is 5).

3.1.4.3 Using the Negotiated Protocol Version

Once a protocol version is negotiated, the session partners SHOULD use in their communication only the protocol elements that are either required or optional for that version (see section [2.2.1](#) for a definition of version-required and version-optional elements) as follows:

- When a partner makes a connection request, it SHOULD use only a connection type that is either required or optional for the negotiated protocol version. If the connection type is optional for the negotiated protocol version, it MUST deal with the other partner not supporting it. [<34>](#)
- When a partner receives a connection request, it SHOULD accept as valid only a connection type that is either required or optional for the negotiated protocol version. Invalid connections SHOULD be rejected, as specified in [\[MS-CMP\]](#) section 2.2.7 (MsgTag 0x00000005). [<35>](#)

- When a partner sends a message over an established connection, it SHOULD use only message types and formats that are supported by the negotiated protocol version in the context of the connection type of the respective connection. [<36>](#)
- When a partner receives a message over an established connection, it SHOULD accept as valid only message types and formats that are supported by the negotiated protocol version in the context of the connection type of the respective connection. An invalid message SHOULD be rejected, as specified in section [3.1.6.<37>](#)

3.1.5 Higher-Layer Triggered Events

None.

3.1.6 Message Processing Events and Sequencing Rules

When an OleTx connection partner receives an incoming message on a connection, it MUST perform the following actions in order to verify the validity of the message:

- Schema validation
 - The Participant MUST validate the message content in accord with the message schema and constraints specified in section [2.2](#) for the specific incoming message type. If a message type is not determinable, the message MUST be considered invalid.
- Connection validation
 - The participant MUST read the connection identifier from the message and look up the corresponding connection object in the session's connection table. If an existing connection object is not found, the message MUST be considered invalid. See section [2.2.4.1](#) for the definition of the MESSAGE_PACKET structure and the **dwConnectionId** and **fIsMaster** fields.
- State validation
 - The participant MUST verify the current state of the connection by using the state field of the connection as follows:
 - If the connection is in the Ended state, the message MUST be considered invalid.
 - If the connection type has not defined a specific processing rule in section [3](#) for the processing of the specific message in the current connection state, then the message MUST be considered invalid.

If an incoming message is considered invalid, the participant MUST ignore the contents of the message. Furthermore, the connection on which the message was received MUST transition to the Ended state, and return a failure result to the higher-layer business logic. The participant MAY also tear down the session with which the connection was established. [<38>](#)

If the connection type defines specific actions that MUST be performed when an invalid message is received, the connection partner MUST also perform those actions. These actions are specified in the Message Processing Events and Sequencing Rules section that specifies the behavior of the connection type.

3.1.7 Timer Events

None.

3.1.8 Other Local Events

An OleTX connection participant **MUST** be able to handle the following events at any time during the lifetime of an OleTX connection.

3.1.8.1 Connection Disconnected

When a connection is disconnected, an OleTX connection participant **MUST**:

- Perform all the actions that are required for a valid disconnection as specified in [\[MS-CMPO\]](#) section **3.2.4.5**.
- If the connection type defines specific additional actions that **MUST** be performed when a connection is disconnected, the OleTx participant **MUST** also perform those actions. These actions are specified in the specific Message Processing Events and Sequencing Rules section that defines the behavior of a specified connection type when receiving incoming messages.
- The connection **MUST** be removed from the connection list that belongs to the transaction that is associated with the connection.
- If the connection state is not already [Ended](#), the state **MUST** be set to Ended.

3.2 Core Transaction Manager Facet Details

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with the behavior that is described in this document.

Note that the abstract data model can be implemented in a variety of ways. This protocol does not prescribe or advocate any specific implementation technique.

The [Core Transaction Manager Facet](#) **MUST** maintain all the data elements in section [3.1.1](#).

The Core Transaction Manager Facet **MUST** also maintain the following data elements:

- **Durable Log:** A durable list of transaction objects. The contents of the log **MUST** persist across software restarts or transient failures.
- **Protocol Extension List:** A list of protocol extensions, as specified in section [3.2.1.6](#).
- **Extended Whereabouts:** A memory buffer that represents the extended whereabouts information of the transaction manager, contributed by protocol extension objects as specified in section [3.2.1.6](#).
- **Extended Whereabouts Size:** The size of the extended whereabouts buffer, in bytes.
- **Extended Whereabouts Protocol Count:** The number of protocol extension objects that contributed to the extended whereabouts information.
- **Security Level:** An enumeration that indicates the security level at which the transaction manager initializes communication by using the [MSDTC Connection Manager: OleTx Transports Protocol Specification](#) and [MSDTC Connection Manager: OleTx Multiplexing Protocol Specification](#)

protocols, as specified in [\[MS-CMPO\]](#) section 3.2.1.1. This element MUST be set to one of the following values:

- **No Security:** This value is set to indicate that the RPC communications MUST NOT require validation of the identity for an incoming message.
- **Incoming Authentication:** This value is set to indicate that the RPC communication SHOULD validate the identity for an incoming message.
- **Mutual Authentication:** This value is set to indicate that the RPC communication SHOULD validate that there is a known identity for an incoming connection. [.<39>](#)
- **Security Flags:** The Core Transaction Manager Facet MUST maintain the following security capability flags:
 - **Allow Network Access:** A flag that indicates whether the transaction manager will communicate with an OleTx participant that is located on a remote machine. If this flag is not set, network access MUST NOT be enabled for the OleTx protocol, regardless of the settings of the other flags.
 - **Allow Network Transactions:** A flag that indicates whether the transaction manager will perform a distributed transaction with an OleTx participant that is located on a remote machine. If the Allow Network Access flag is set to false, this flag MUST be ignored.
 - **Allow Inbound Transactions:** A flag that indicates whether the transaction manager will act as subordinate to a superior transaction manager facet that is located on a remote machine. If either the Allow Network Access flag or the Allow Network Transactions flags are set to false, this flag MUST be ignored.
 - **Allow Outbound Transactions:** A flag that indicates whether the transaction manager will act as superior to a [subordinate transaction manager facet](#) that is located on a remote machine. If either the Allow Network Access flag or the Allow Network Transactions flag is set to false, this flag MUST be ignored.
 - **Allow Remote Administration:** A flag that indicates whether the transaction manager will be administered by an application that is located on a remote machine. If the Allow Network Access flag is set to false, this flag MUST be ignored.
 - **Allow Remote Clients:** A flag that indicates whether the transaction manager will communicate with an application or a resource manager that is located on a remote machine. If the Allow Network Access flag is set to false, this flag MUST be ignored.
 - **Allow TIP:** A flag that indicates whether the transaction manager has enabled the TIP protocol, as specified in [\[RFC2371\]](#). For information on the transaction manager's interaction with [\[RFC2371\]](#), see [\[MS-DTCM\]](#). If the Allow Network Access flag is set to false, this flag MUST be ignored.
 - **Allow XA:** A flag that indicates whether the transaction manager provides support for the [\[C193\]](#) protocol in an implementation-specific manner.

The Core Transaction Manager Facet MUST extend the definition of a transaction object to include the following data elements:

- **Superior Enlistment:** A reference to an Enlistment object that belongs to either the subordinate transaction manager facet or the transaction manager communicating with an application facet, as specified in [Enlistment Objects \(section 3.2.1.3\)](#).

- **Next Phase Zero Wave Enlistment list:** A list of Enlistment objects that represent the set of Phase Zero enlistments that belong to the next Phase Zero wave of the transaction.
- **Phase Zero Enlistment list:** A list of Enlistment objects that represent the set of Phase Zero enlistments that belong to the current Phase Zero wave of the transaction.
- **Phase One Enlistment list:** A list of Enlistment objects that represent the set of Phase One enlistments currently registered on the transaction.
- **Phase One Voter Enlistment list:** A list of Enlistment objects that represent the set of voter enlistments currently registered on the transaction.
- **Phase Two Enlistment list:** A list of Enlistment objects that represent the set of Phase One enlistments who voted Prepared when asked to vote on the outcome of the transaction.
- **Phase Two Voter Enlistment list:** A list of Enlistment objects that represent the set of voter enlistments who voted Prepared when asked to vote on the outcome of the transaction.
- **Root:** A flag set to true if the Core Transaction Manager Facet is the root of the transaction; otherwise, false.
- **Doomed:** A flag set to true if the transaction has been aborted; otherwise, false.
- **Attributes Set:** A flag set to true when the transaction attributes are updated by using the Set Transaction Attributes event.
- **Phase Zero Registered:** A flag set to true if the transaction has successfully registered for the next Phase Zero wave; otherwise, false.
- **Single Phase Commit:** A flag set to true if the Core Transaction Manager Facet was requested to perform a Single Phase Commit on the transaction; otherwise, false.
- **State:** A State enumeration that represents the current state of the transaction. These states are as specified in section [3.2.1.4](#).
- **Isolation Level:** An Isolation Level value.
- **Isolation Flags:** An Isolation Flags value.
- **Description:** An implementation-specific description string that is provided to the core transaction manager when the transaction is created.
- **Timeout:** A 32-bit unsigned integer that represents the number of milliseconds after which a root transaction MUST time out if an outcome is not reached. This value MUST be used to initialize the [Transaction Timeout timer \(section 3.2.2.1\)](#).

The Core Transaction Manager Facet MUST extend the definition of a connection object, as specified in [MS-CMP] section 3.1.1.1, to include the following data elements:

- **Enlistment:** A reference to the Enlistment object that is associated with the connection. Some connections do not use this field.
- **Connection-Specific Data:** An opaque reference to an object. This field is used during the execution of a connection to associate connection-specific objects with the connection. Some connections do not use this field.

3.2.1.1 Versioning

The core transaction manager MUST maintain the data that pertains to the extended whereabouts functionality only on versions where the connection type [CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS](#) is supported as specified in [2.2.8](#). The following data elements, as specified in [3.2.1](#), are affected:

- Extended Whereabouts.
- Extended Whereabouts Size.
- Extended Whereabouts Protocol Count.
- Extended Whereabouts data structures that are provided by protocol extension objects:
 - **Whereabouts**.
 - Whereabouts Size.

The core transaction manager MUST maintain the data that pertains to the [Phase Zero](#) functionality only on versions where the connection type [CONNTYPE_TXUSER_PHASE0](#) is supported as specified in [2.2.10](#). The following data elements, as specified in [3.2.1](#), are affected:

- Next Phase Zero WaveEnlistment list.
- Phase Zero Enlistment list.
- Phase Zero Registered.

3.2.1.2 Transaction Logging

When a transaction object is stored in the durable log of the [Core Transaction Manager Facet](#), the Core Transaction Manager Facet MUST record only the following fields:

- The **Identifier** field.
- The **State** field. When a transaction object is stored in the durable log, this field MUST be set to one of the following two states:
 - [In Doubt](#)
 - [Failed to Notify](#)
- The **Phase Two Enlistment** list.
- If the **State** field of the transaction is set to In Doubt, the **Superior Enlistment** field MUST be stored.

When an Enlistment object is stored in the durable log of the Core Transaction Manager Facet, the Core Transaction Manager Facet MUST record all the object fields.

When a connection object is stored in the durable log of the Core Transaction Manager Facet, the Core Transaction Manager Facet MUST record all the object fields.

When a connection object is retrieved from the durable log of the Core Transaction Manager Facet, its state MUST be set to [Ended](#).

3.2.1.3 Enlistment Objects

An Enlistment Object MUST contain the following data structures:

- **Transaction Manager Facet:** A reference to the specific facet in the transaction manager that created the Enlistment object. A single facet creates zero or more Enlistment objects. Transaction manager facets are as specified in section [3.2.1.5](#).
- **Transaction:** A reference to a transaction object.
- **Connection:** A reference to a connection object.
- **Resource Manager Identifier:** A reference to a resource manager identifier. This field MUST be set if the Enlistment object belongs to the transaction manager communicating with a resource manager facet.
- **Recovery Information:** An extensibility point that allows transaction manager facets to contribute information to the durable log that is returned to them when recovery occurs. This field MUST only be interpreted by the transaction manager facet that created the Enlistment object.
- **Name:** A string providing a name for the enlistment. Each transaction manager facet MUST define the contents of this field for the Enlistment objects that are created by that facet.
- **Identifier:** A string providing an identifier for the enlistment. Each transaction manager facet MUST provide the contents of this field for Enlistment objects that are created by that facet.

3.2.1.4 Transaction States

The state field of the transaction object MUST represent the set of different states to which the logical state machine of the transaction MUST be set.

The transaction state machine MUST support the following states:

- [Idle](#)
- [Active](#)
- [Phase Zero](#)
- [Phase Zero Complete](#)
- [Voting](#)
- [Voting Complete](#)
- [Phase One](#)
- [Phase One Complete](#)
- [Single Phase Commit](#)
- [Committing](#)
- [Aborting](#)
- [In Doubt](#)

- [Failed to Notify](#)
- [Ended](#)

The following diagram reflects states and the events that directly change them. The transaction manager and the transaction can receive more events than those shown, but those events do not affect the state of the transaction.

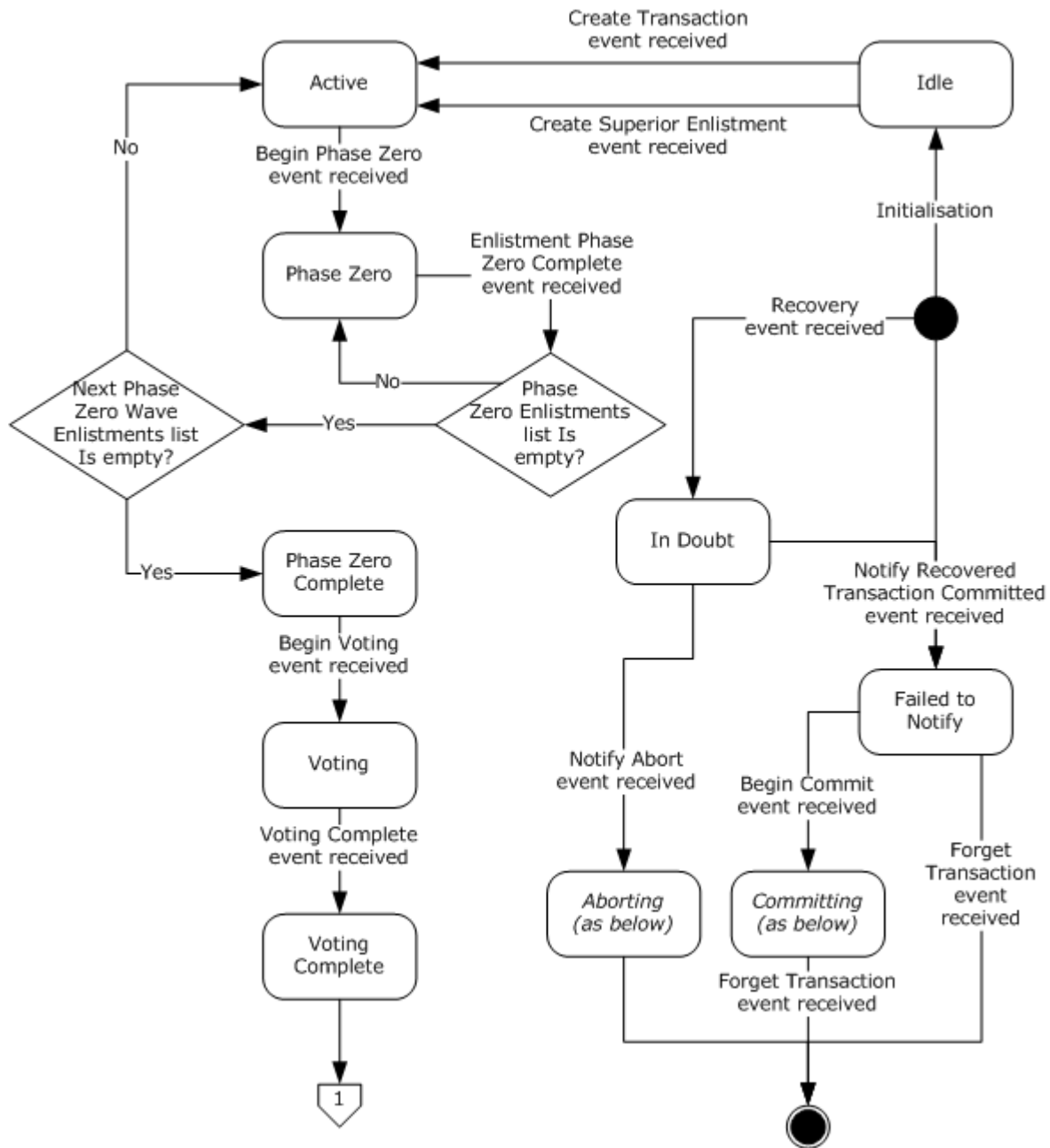


Figure 12: Transaction manager states and events (Phase Zero)

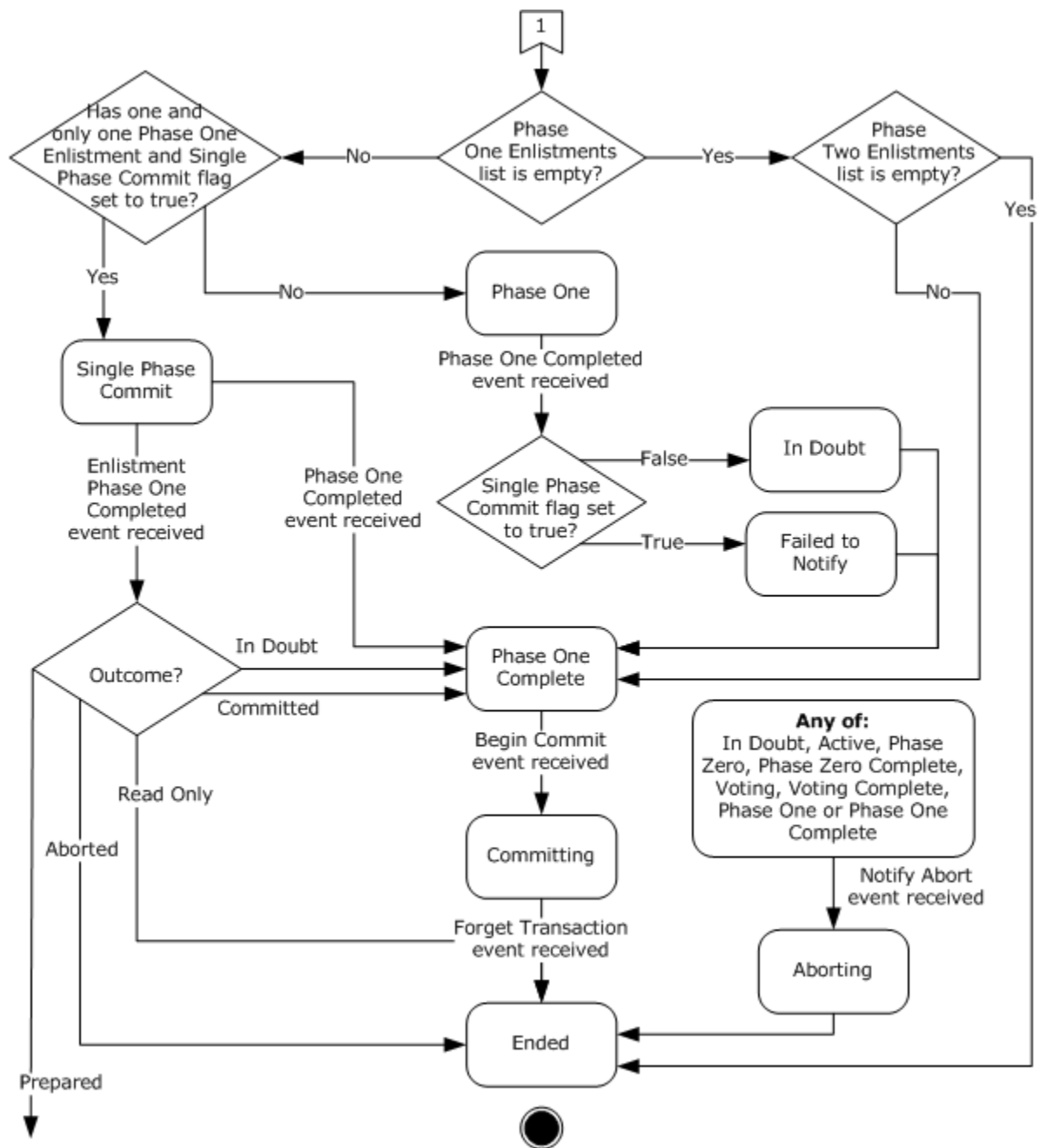


Figure 13: Transaction manager states and events (Phase One)

3.2.1.4.1 Idle

This is the initial state. The following events are processed in this state:

- [Create Transaction](#)
- [Create Superior Enlistment](#)

- [Associate Transaction](#)
- [Branch Transaction Success](#)
- [Branch Transaction Failure](#)

3.2.1.4.2 Active

The following events are processed in this state:

- [Create Phase Zero Enlistment](#)
- [Create Voter Enlistment](#)
- [Create Subordinate Enlistment](#)
- [Register Phase Zero Success](#)
- [Register Phase Zero Failure](#)
- [Export Transaction](#)
- [Set Transaction Attributes](#)
- [Set Transaction Timeout](#)
- [Begin Phase Zero](#)
- [Enlistment Unilaterally Aborted](#)
- [Notify Aborted](#)
- [Unenlist Phase Zero Enlistment](#)
- [Transaction Timeout Timer](#)

3.2.1.4.3 Phase Zero

The following events are processed in this state:

- [Create Phase Zero Enlistment](#)
- [Create Voter Enlistment](#)
- [Create Subordinate Enlistment](#)
- [Register Phase Zero Success](#)
- [Register Phase Zero Failure](#)
- [Export Transaction](#)
- [Set Transaction Timeout](#)
- [Enlistment Phase Zero Complete](#)
- [Enlistment Unilaterally Aborted](#)
- [Notify Aborted](#)

- [Unenlist Phase Zero Enlistment](#)
- [Transaction Timeout Timer](#)

3.2.1.4.4 Phase Zero Complete

The following events are processed in this state:

- [Create Phase Zero Enlistment](#)
- [Create Voter Enlistment](#)
- [Create Subordinate Enlistment](#)
- [Register Phase Zero Success](#)
- [Register Phase Zero Failure](#)
- [Export Transaction](#)
- [Set Transaction Timeout](#)
- [Begin Phase One](#)
- [Begin Voting](#)
- [Enlistment Unilaterally Aborted](#)
- [Notify Aborted](#)
- [Transaction Timeout Timer](#)

3.2.1.4.5 Voting

The following events are processed in this state:

- [Set Transaction Timeout](#)
- [Enlistment Vote Complete](#)
- [Voting Complete](#)
- [Enlistment Unilaterally Aborted](#)
- [Notify Aborted](#)
- [Transaction Timeout Timer](#)

3.2.1.4.6 Voting Complete

The following events are processed in this state:

- [Set Transaction Timeout](#)
- [Begin Commit](#)
- [Enlistment Unilaterally Aborted](#)
- [Notify Aborted](#)

- [Forget Transaction](#)
- [Transaction Timeout Timer](#)

3.2.1.4.7 Phase One

The following events are processed in this state:

- [Set Transaction Timeout](#)
- [Enlistment Vote Complete](#)
- [Enlistment Phase One Complete](#)
- [Enlistment Unilaterally Aborted](#)
- [Notify Aborted](#)
- [Phase One Completed](#)
- [Transaction Timeout Timer](#)

3.2.1.4.8 Phase One Complete

The following events are processed in the Phase One Complete state:

- [Begin Commit](#)
- [Begin In Doubt](#)
- [Forget Transaction](#)

3.2.1.4.9 Single Phase Commit

The following events are processed in this state:

- [Enlistment Phase One Complete](#)
- [Phase One Completed](#)

3.2.1.4.10 Committing

The following events are processed in this state:

- [Begin Commit](#)
- [Enlistment Commit Complete](#)
- [Forget Transaction](#)
- [Request Transaction Outcome](#)

3.2.1.4.11 Aborting

The following events are processed in the Aborting state:

- [Begin Rollback](#)

- [Enlistment Rollback Complete](#)
- [Forget Transaction](#)
- [Request Transaction Outcome](#)

3.2.1.4.12 In Doubt

The following events are processed in the In Doubt state:

- [Notify Recovered Transaction Committed](#)
- [Forget Transaction](#)
- [Resolve Transaction](#)
- [Recovery Event](#)

3.2.1.4.13 Failed to Notify

The following events are processed in the Failed to Notify state.

- [Begin Commit](#)
- [Notify Recovered Transaction Committed](#)
- [Forget Transaction](#)
- [Resolve Transaction](#)
- [Request Transaction Outcome](#)
- [Recovery Event](#)

3.2.1.4.14 Ended

This is the final state. The following event is processed in this state:

- [Request Transaction Outcome](#)

3.2.1.5 Transaction Manager Facets

An OleTX transaction manager is subdivided into the following transaction manager facets:

- [Core Transaction Manager Facet.](#)
- [Transaction manager communicating with an application facet.](#)
- [Transaction manager communicating with a resource manager facet.](#)
- [Superior transaction manager facet.](#)
- [Subordinate transaction manager facet.](#)

These facets MUST NOT provide partial failure modes. Either all facets MUST be in an available state, or all facets MUST be in a failed state.

These facets MUST communicate with each other by using a set of events. Each facet MUST define the set of events that the facet supports.

An event MUST consist of the following data elements:

- The name of the event.
- The list of arguments with which the event MUST be signaled.

This protocol assumes the existence of an implementation-specific communication mechanism used to signal events between facets inside a transaction manager. This communication mechanism MUST NOT allow man-in-the-middle or other classes of intermediary attacks. <40>

Each facet MUST provide a definition for the **Name** and **Identifier** fields of an [Enlistment object](#), as specified in section [3.2.1.3](#).

The conceptual model that is described here requires that one and only one thread of operation be active inside the facets that make up the [transaction manager](#).

3.2.1.6 Protocol Extension Objects

A protocol extension is an implementation-specific module that represents the ability to perform transaction processing by using a transaction coordination protocol that is not OleTx.

Protocol extension objects MUST leverage the following vendor extensibility points in the [Core Transaction Manager Facet](#):

- The ability to augment the list of transaction manager facets, as specified in section [3.2.1.5](#), to include additional protocol-specific facets.
- The ability to define custom behavior for the **Name** and **Property** fields on enlistment objects that are created inside these facets.
- The ability to communicate with the core transaction manager by using the events that are specified in section [3.2.4](#).
- The ability to contribute whereabouts information to the **extended whereabouts** field of the core transaction manager, as specified in section [3.3.5.2.2.1](#).
- The ability to contribute recovery information to enlistment objects that are stored in the durable log, as specified in section [3.2.1.3](#).

A protocol extension object MUST provide the following data structures:

- **Identifier**: A GUID that uniquely identifies the protocol extension.
- **Whereabouts**: An array of bytes that represents the protocol extension.
- **Whereabouts Size**: The size of the Whereabouts array.

3.2.2 Timers

The [Core Transaction Manager Facet](#) MUST provide a [transaction time-out timer](#).

3.2.2.1 Transaction Time-out Timer

This timer MUST be set when a new transaction is created. It MUST be canceled when a transaction enters one of the following states:

- [Phase One Complete](#)
- [Single Phase Commit](#)
- [Committing](#)
- [Aborting](#)
- [Ended](#)

The default value is specified by the **Timeout** field on the transaction object for which the instance of the timer is set. The minimum value of the timer MUST be zero, which means that the timer never generates a timer event. [<41>](#)

When the timer is initialized, the initialization MUST provide a transaction object to associate with the timer. When the timer expires, the same transaction object MUST be provided alongside the timer notification. The [Core Transaction Manager Facet](#) MUST provide a distinct transaction time-out timer instance for each active transaction. If an implementation sets the value of the timeout timer associated with a transaction object to zero, the Transaction Timeout Timer event (see [3.2.6.1](#)) is never signaled, and therefore the transaction never times out (for the negative consequences of this behavior see [GRAY]).

3.2.3 Initialization

When the [Core Transaction Manager Facet](#) is initialized:

- The **Security Level** field MUST be set to a value that is obtained from an implementation-specific source.
- The following security flags MUST be set to a value that is obtained from an implementation-specific source:
 - Allow Network Access
 - Allow Network Transactions
 - Allow Inbound Transactions
 - Allow Outbound Transactions
 - Allow Remote Administration
 - Allow Remote Clients
 - Allow TIP
 - Allow XA
- If the Allow Network Access flag is set to false:
 - The transport protocol MUST be initialized by using the following values:
 - The **Security Level** field, as specified in [\[MS-CMPO\]](#) section 3.2.1.1.

- The `PROT_LRPC` flag, as specified in [MS-CMPO], section [2.2.7](#).
- Otherwise:
 - The transport protocol **MUST** be initialized by using the following values:
 - The **Security Level** field, as specified in [\[MS-CMPO\]](#) section 3.2.1.1.
 - A bitwise OR operator of one or more flags obtained from an implementation-specific source, as specified in [MS-CMPO], section 2.2.7.
 - The protocol extension list **MUST** be populated with instances that are obtained from an implementation-specific source.
 - If the protocol extension list is not empty, the Core Transaction Manager Facet **MUST** perform the following actions:
 - Create a temporary list of protocol extension objects.
 - Create a total size integer, initialized to zero.
 - For each protocol extension object in the protocol extension list of the Core Transaction Manager Facet:
 - If the **Whereabouts Size** field of the object is not zero:
 - Add the object to the temporary list.
 - Add the **Whereabouts Size** field of the object that is aligned on 4 to the total size.
 - Allocate a memory buffer whose size in bytes is equal to the total size.
 - For each protocol extension object in the temporary list, write the following bytes to the memory buffer:
 - Four bytes set to the value of `TmProtocolExtended`, as specified in [2.2.5.9](#).
 - Four bytes set to the value of the protocol extension object's **Whereabouts Size** field plus sixteen.
 - Sixteen bytes set to the byte representation of the protocol extension object's **Identifier** field.
 - A number of bytes whose size is equal to the size of the protocol extension object's whereabouts array set to the data in the protocol extension object's **Whereabouts** field.
 - A sufficient number of bytes (each byte set to zero) in order to pad the end of the **Whereabouts** field to a four-byte alignment.
 - Assign the memory buffer to the **Extended Whereabouts** field of the core transaction manager.
 - Assign the size of the memory buffer to the **Extended Whereabouts Size** field of the Core Transaction Manager Facet.
 - Assign the number of elements in the temporary list to the **Extended Whereabouts Protocol Count** field of the Core Transaction Manager Facet.

3.2.3.1 Core Transaction Manager Facet Initialization

The [Core Transaction Manager Facet](#) MUST perform the following initialization steps when the facet is initialized:

- The **Security Level** field MUST be set to an implementation-specific value. [<42>](#)
- The following security flags MUST be set to an implementation-specific value: [<43>](#)
 - Allow Network Access
 - Allow Network Transactions
 - Allow Inbound Transactions
 - Allow Outbound Transactions
 - Allow Remote Administration
 - Allow Remote Clients
 - Allow TIP
 - Allow XA
- If the Allow Network Access flag is set to false:
 - The transport protocol MUST be initialized with the following values:
 - A security-level value as specified in [\[MS-CMPO\]](#) section 3.2.1.1.
 - The PROT_LRPC flag as specified in [\[MS-CMPO\]](#), section [2.2.7](#).
- Otherwise:
 - The transport protocol MUST be initialized with the following values:
 - A security-level value, as specified in [\[MS-CMPO\]](#) section 3.2.1.1.
 - An enumeration value that is obtained from an implementation-specific source, as specified in [\[MS-CMPO\]](#), section 2.2.7.
- The protocol extension list MUST be populated with instances that are obtained in an implementation-specific manner.
- If the protocol extension list is not empty, the Core Transaction Manager Facet MUST perform the following actions:
 - Query each protocol extension for its extended whereabouts information by using the **Whereabouts** and **Whereabouts Size** fields of the object.
 - Create an array of [STmToTmProtocol](#) structures and assign it to the **Extended Whereabouts** field of the Core Transaction Manager Facet:
 - The array MUST contain an entry for each protocol extension that contributes extended whereabouts information.
 - The **tmprotDescribed** field of each entry MUST be set to TmProtocolExtended, as specified in section [2.2.5.9](#).

- The **rgbTmProtocolData** field of each entry MUST contain an [SExtendedEndpointInfo](#) structure, as specified in section [2.2.5.8](#).
- Assign the size, in bytes, of the STmToTmProtocol array to the **Extended Whereabouts Size** field of the Core Transaction Manager Facet.
- Assign the number of protocol extensions that contribute extended whereabouts information to the STmToTmProtocol array to the **Extended Whereabouts Protocol Count** field of the Core Transaction Manager Facet.

3.2.3.2 Transaction Object Initialization

The [Core Transaction Manager Facet](#) MUST initialize each new transaction object that is created by the facet with the following default values:

- The **Root** field MUST default to false.
- The **Doomed** field MUST default to false.
- The **Attributes Set** field MUST default to false.
- The **Phase Zero Registered** field MUST be set to false.
- The **Single Phase Commit** field MUST default to false.
- The **State** MUST default to Idle.
- The **Isolation Level** field MUST default to Serializable.
- The **Isolation Flags** field MUST default to 0.
- The **Description** field MUST default to an empty string.
- The Timeout value MUST [<44>](#) default to a value that is obtained in an implementation-specific manner.

3.2.4 Higher-Layer Triggered Events

The operation of the [Core Transaction Manager Facet](#) MUST be prepared to process the higher-layer events in this section.

3.2.4.1 Recovery Event

This event is triggered by the higher-layer software hosting infrastructure when it reinitializes the system after a software failure or restart.

When the [Core Transaction Manager Facet](#) is asked to recover after a software failure or restart, it MUST perform the following actions:

- For each transaction object in the durable log of the Core Transaction Manager Facet:
 - Copy the transaction object to the transaction table of the Core Transaction Manager Facet.
- For each transaction object in the transaction table of the Core Transaction Manager Facet:
 - If the transaction state is [In Doubt \(section 3.2.1.4.12\)](#):

- Signal the [Recover In Doubt Transaction \(section 3.8.7.8\)](#) event on the transaction manager facet that is referenced by the transaction object's **Superior Enlistment** field with the value of the transaction object's **Superior Enlistment** field.
- Otherwise:
 - Signal the [Notify Recovered Transaction Committed \(section 3.2.7.24\)](#) event on the Core Transaction Manager Facet with the transaction object.

All incoming connections MUST be rejected until all transactions in the durable log are copied to the transaction table. Connection rejection is as specified in [\[MS-CMPO\]](#), section [3.1.4](#), and [\[MS-CMP\]](#), section [3.1.5.5](#).

3.2.5 Message Processing Events and Sequencing Rules

3.2.6 Timer Events

3.2.6.1 Transaction Timeout Timer

When this timer expires, the core transaction manager MUST perform the following actions:

- If the provided transaction object is in one of the following states, the core transaction manager MUST ignore the timer event.
 - [Phase One Complete](#)
 - [Single Phase Commit](#)
 - [Committing](#)
 - [Aborting](#)
 - [In Doubt](#)
 - [Failed to Notify](#)
 - [Ended](#)
- Otherwise, the core transaction manager MUST:
 - Signal the [Unilaterally Aborted](#) event on the transaction's Superior Enlistment's transaction manager facet with the Superior Enlistment object of the transaction.
 - Signal the [Notify Aborted](#) event on the [Core Transaction Manager Facet](#) using the provided transaction object.

3.2.7 Other Local Events

The core transaction manager MUST be prepared to process the local events that are defined in the following sections.

If the [Core Transaction Manager Facet](#) supports the [CONNTYPE_TXUSER_PHASE0](#) connection type, this facet MUST be prepared to process local events that pertain to [Phase Zero](#) functionality. The following local events are affected:

- [Create Phase Zero Enlistment](#)

- [Register Phase Zero Success](#)
- [Register Phase Zero Failure](#)
- [Begin Phase Zero](#)
- [Enlistment Phase Zero Complete](#)
- [Unenlist Phase Zero Enlistment](#)

3.2.7.1 Associate Transaction

The Associate Transaction event MUST be signaled with the following arguments:

- A transaction object.
- A Name object representing the remote superior transaction manager.

If the Associate Transaction event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- If the durable log is too full to accept a new enlistment:
 - Signal the [Associate Transaction Failure \(section 3.4.7.1\)](#) event on the [Transaction Manager Communicating with an Application Facet \(section 1.3.3.3.2\)](#) with the following arguments:
 - The provided transaction object.
 - The Log Full Local reason code.
- Otherwise:
 - Signal the [Branch Transaction \(section 3.8.7.1\)](#) event on the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) with the following arguments:
 - The provided transaction object.
 - The provided name object.

3.2.7.2 Begin Commit

The Begin Commit event MUST be signaled with the following arguments:

- A transaction object.

If the Begin Commit event is signaled, the [Core Transaction Manager Facet](#) MUST perform the following actions:

- Set the transaction state to [Committing \(section 3.2.1.4.10\)](#).
- If the [Phase Two](#) Voter list of the transaction is not empty:
 - For each Enlistment object in the Phase Two Voter enlistment list of the transaction:
 - Remove the Enlistment object from the Phase Two Voter enlistment list of the transaction.
 - Signal the Begin Commit event (see sections [3.4.7.3](#), [3.6.7.1](#), and [3.7.7.1](#)) on the enlistment's transaction manager facet field with the Enlistment object.

- If the Phase Two Enlistment list of the transaction is not empty:
 - For each Enlistment object in the Phase Two Enlistment list of the transaction:
 - Signal the Begin Commit event (see sections [3.4.7.3](#), [3.6.7.1](#), and [3.7.7.1](#)) on the enlistment's transaction manager facet field with the Enlistment object.
- Otherwise, if the Phase Two Enlistment list of the transaction is empty:
 - Signal the [Commit Complete \(section 3.8.7.3\)](#) event on the transaction's superior enlistment's transaction manager facet with the transaction's superior Enlistment object.
 - Signal the [Forget Transaction \(section 3.2.7.22\)](#) event on the Core Transaction Manager Facet with the provided transaction object.

3.2.7.3 Begin In Doubt

The Begin In Doubt event MUST be signaled with the following arguments:

- A transaction object.

If the Begin In Doubt event is signaled, the [Core Transaction Manager Facet](#) MUST perform the following actions:

- For each Enlistment object in the [Phase Two](#) Voter Enlistment list of the transaction:
 - Signal the Begin In Doubt event (see sections [3.4.7.4](#) and [3.6.7.2](#)) on the Enlistment object's transaction manager facet with the Enlistment object.
- Signal the [Forget Transaction \(section 3.2.7.22\)](#) event on the Core Transaction Manager Facet with the transaction object of the enlistment.

3.2.7.4 Begin Phase One

The Begin Phase One event MUST be signaled with the following arguments:

- A transaction object.
- A flag indicating whether the transaction SHOULD or MUST NOT attempt to perform a single-phase commit.

If the Begin Phase One event is signaled, the [Core Transaction Manager Facet](#) MUST perform the following actions:

- Set the **Single Phase Commit** field of the transaction to the value of the provided Single Phase Commit flag.
- Signal the [Begin Voting \(section 3.2.7.7\)](#) event on the Core Transaction Manager Facet with the following argument:
 - The provided transaction object.

3.2.7.5 Begin Phase Zero

The Begin Phase Zero event MUST be signaled with the following arguments:

- A transaction object.

If the Begin Phase Zero event is signaled, the [Core Transaction Manager Facet](#) MUST perform the following actions:

- Set the transaction state to [Phase Zero \(section 3.2.1.4.3\)](#).
- Move each enlistment object in the Next Phase Zero Wave Enlistment list of the transaction to the Phase Zero Enlistment list of the transaction.
- Set the Phase Zero Registered flag of the transaction object to false.
- If the Phase Zero Enlistment list of the transaction is not empty:
 - For each enlistment object in the Phase Zero Enlistment list of the transaction:
 - Signal the Begin Phase Zero event (see sections [3.6.7.4](#) and [3.7.7.3](#)) on the enlistment object's transaction manager facet with the enlistment object.
- Otherwise:
 - Set the transaction state to [Phase Zero Complete \(section 3.2.1.4.4\)](#).
 - Signal the [Phase Zero Complete \(section 3.4.7.14\)](#) event on the superior enlistment's transaction manager facet of the transaction with the following arguments:
 - The superior enlistment object of the transaction.
 - The success outcome.

3.2.7.6 Begin Rollback

The Begin Rollback event MUST be signaled with the following arguments:

- A transaction object.

If the Begin Rollback event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Signal the [Rollback Complete \(section 3.4.7.17\)](#) event on the transaction's superior enlistment's transaction manager facet with the superior enlistment object of the transaction.
- Signal the [Notify Aborted \(section 3.2.7.23\)](#) event on the Core Transaction Manager Facet with the provided transaction object.

3.2.7.7 Begin Voting

The Begin Voting event MUST be signaled with the following arguments:

- A transaction object.

If the Begin Voting event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Set the transaction state to [Voting \(section 3.2.1.4.5\)](#).
- If the [Phase One \(section 1.3.1.2\)](#) Voter Enlistment list of the transaction is empty:
 - Signal the [Voting Complete \(section 3.2.7.35\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the provided transaction object.

- Otherwise:
 - For each enlistment object in the Phase One (section 1.3.1.2) Voter Enlistment list of the transaction:
 - Signal the Begin Voting event (see sections [3.4.7.6](#) and [3.6.7.6](#)) on the enlistment's transaction manager facet field with the enlistment object.
 - If the Phase One (section 1.3.1.2) Enlistment list of the transaction contains more than one element, or if it contains one element and the [Single Phase Commit \(section 1.3.2.2\)](#) flag of the transaction is set to false:
 - For each enlistment object in the Phase One (section 1.3.1.2) Enlistment list of the transaction:
 - Signal the [Begin Phase One \(section 3.7.7.2\)](#) event on the enlistment's transaction manager facet field with the following argument:
 - The enlistment object.
 - The Single Phase Commit (section 1.3.2.2) flag set to false.

3.2.7.8 Branch Transaction Failure

The Branch Transaction Failure event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:
 - Log Full Remote
 - No Mem
 - Too Late
 - Too Many Remote
 - Tx Not Found
 - Comm Failed

If the Branch Transaction Failure (section 3.2.7.8) event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Signal the [Associate Transaction Failure \(section 3.4.7.1\)](#) event on the transaction manager communicating with an application facet with the following arguments:
 - The provided transaction object.
 - The provided reason code.

3.2.7.9 Branch Transaction Success

The Branch Transaction Success event MUST be signaled with the following arguments:

- An Enlistment object.

If the Branch Transaction Success event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Add the transaction object of the enlistment to the transaction table of the Core Transaction Manager Facet (section 1.3.3.3.1).
- Set the superior enlistment of the transaction to the provided enlistment object.
- Signal the [Associate Transaction Success \(section 3.4.7.2\)](#) event on the transaction manager communicating with an application facet with the transaction object of the enlistment.

3.2.7.10 Create Phase Zero Enlistment

The Create Phase Zero Enlistment event MUST be signaled with the following arguments:

- An Enlistment object.

If the Create Phase Zero Enlistment event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- If the provided transaction state of the enlistment is [Phase Zero \(section 3.2.1.4.3\)](#):
 - The Core Transaction Manager Facet (section 1.3.3.3.1) MUST:
 - Signal the [Create Phase Zero Enlistment Success \(section 3.6.7.8\)](#) event on the enlistment object's transaction manager facet with the provided enlistment object.
 - Signal the [Begin Phase Zero \(section 3.2.7.5\)](#) event on the provided enlistment object's transaction manager facet with the provided enlistment object.
- Otherwise, if the transaction state is [Active \(section 3.2.1.4.2\)](#) or [Phase Zero Complete \(section 3.2.1.4.4\)](#):
 - If the Next Phase Zero Wave Enlistment list of the transaction is empty:
 - Signal the [Register Phase Zero \(section 3.4.7.15\)](#) event on the transaction's superior enlistment's transaction manager facet with the superior enlistment object of the provided transaction.
 - Otherwise, if the list is non-empty and the Phase Zero Registered flag of the transaction is true:
 - Signal the Create Phase Zero Enlistment Success (section 3.6.7.8) event on the enlistment object's transaction manager facet with the enlistment object.
 - Add the provided enlistment to the Next Phase Zero Wave Enlistment list of the transaction.
- Otherwise:
 - Signal the [Create Phase Zero Enlistment Failure \(section 3.6.7.7\)](#) event on the enlistment object's transaction manager facet field with the following arguments:
 - The provided enlistment object.
 - The Too Late reason code.

3.2.7.11 Create Subordinate Enlistment

The Create Subordinate Enlistment event MUST be signaled with the following arguments:

- An Enlistment object.

If the Create Subordinate Enlistment event is signaled, the [Core Transaction Manager Facet](#) MUST perform the following actions:

- If the transaction state is not either [Active \(section 3.2.1.4.2\)](#) or [Phase Zero \(section 3.2.1.4.3\)](#) or [Phase Zero Complete \(section 3.2.1.4.4\)](#):
 - Signal the [Create Subordinate Enlistment Failure \(section 3.6.7.10\)](#) event on the Enlistment object's Transaction Manager Facet with the following arguments:
 - The provided Enlistment object.
 - The Too Late reason code.
- Otherwise, if the Durable Log is too full to accept a new enlistment:
 - Signal the Create Subordinate Enlistment Failure (section 3.6.7.10) event on the Enlistment object's Transaction Manager Facet with the following arguments:
 - The provided Enlistment object.
 - The Log Full reason code.
- Otherwise, if the implementation-specific limit on Subordinate enlistments is exceeded: [<45>](#)
 - Signal Create Subordinate Enlistment Failure (section 3.6.7.10) event on the Enlistment object's Transaction Manager Facet with the following arguments:
 - The provided Enlistment object.
 - The Too Many reason code.
- Otherwise:
 - Add the provided Enlistment object to the transaction's Phase One Enlistment list.
 - Signal the [Create Subordinate Enlistment Success \(section 3.6.7.11\)](#) event on the Enlistment object's Transaction ManagerFacet with the provided Enlistment object.

3.2.7.12 Create Superior Enlistment

The Create Superior Enlistment event MUST be signaled with the following arguments:

- An Enlistment object.

If the Create Superior Enlistment event is signaled, the Core Transaction Manager MUST perform the following actions:

- If the transaction referenced by the provided Enlistment object already exists in the transaction table:
 - Signal the [Create Superior Enlistment Failure \(section 3.8.7.5\)](#) event on the Transaction Manager Facet referenced by the provided Enlistment object with the following arguments:

- The provided Enlistment object.
- The Duplicate reason code.
- Otherwise, if the Durable Log is too full to accept a new enlistment:
 - Signal the Create Superior Enlistment Failure (section 3.8.7.5) event on the Enlistment object's Transaction Manager Facet with the following arguments:
 - The provided Enlistment object.
 - The Log Full reason code.
- Otherwise:
 - Add the Transaction object referenced by the provided Enlistment object to the Transaction Table, using the Transaction's Identifier as the key.
 - Set the transaction's Superior Enlistment field to the provided Enlistment object.
 - Set transaction state to Active.
 - Set the transaction's Root flag to false.
 - Signal the [Create Superior Enlistment Success \(section 3.8.7.4\)](#) event on the Transaction Manager Facet referenced by the provided Enlistment object with the provided Enlistment object.

3.2.7.13 Create Transaction

The Create Transaction event MUST be signaled with the following arguments:

- An Enlistment object

If the Create Transaction event is signaled, the Core Transaction Manager MUST perform the following actions:

- The Core Transaction Manager SHOULD:

[<46>](#)

- Look for an existing entry in the transaction table, using the identifier of the Transaction object referenced by the provided Enlistment object as the key.
- If an entry exists:
 - Signal the [Create Transaction Failure \(section 3.4.7.7\)](#) event on the Transaction Manager Facet referenced by the provided Enlistment object with the following arguments:
 - The provided Transaction object.
 - The Duplicate reason code.
 - Cease processing the event.
- Add the Transaction object referenced by the provided Enlistment object to the transaction table, using the Transaction's identifier as the key.
- Set the transaction's Superior Enlistment to the provided Enlistment object.

- Set the transaction's Root flag to true.
- Set transaction's state to Active.
- Initialize the Transaction Timeout timer with the following arguments:
 - The Transaction object.
 - The Transaction object's Timeout value.
- Signal the [Create Transaction Success \(section 3.4.7.8\)](#) event on the Transaction Manager Facet referenced by the provided Enlistment with the Transaction object referenced by the provided Enlistment object.

3.2.7.14 Create Voter Enlistment

The Create Voter Enlistment event MUST be signaled with the following arguments:

- An Enlistment object.

If the Create Voter Enlistment event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- If the transaction state is not either [Active \(section 3.2.1.4.2\)](#) or [Phase Zero \(section 3.2.1.4.3\)](#) or [Phase Zero Complete \(section 3.2.1.4.4\)](#):
 - Signal the [Create Voter Enlistment Failure \(section 3.4.7.9\)](#) event on the Enlistment object's Transaction Manager Facet with the following arguments:
 - The Enlistment object.
 - The Too Late reason code.
- Otherwise:
 - Add the provided enlistment to the transaction's Phase One Voter Enlistment list.
 - Signal the [Create Voter Enlistment Success \(section 3.4.7.10\)](#) event on the Enlistment object's Transaction Manager Facet with the provided Enlistment object.

3.2.7.15 Enlistment Commit Complete

The Enlistment Commit Complete event MUST be signaled with an Enlistment object.

If the Enlistment Commit Complete event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Remove the enlistment from the transaction's Phase Two Enlistment list.
- If the transaction's Phase Two Enlistment list is now empty:
 - If the transaction's Single Phase Commit flag is false and the transaction state is not [Failed to Notify \(section 3.2.1.4.13\)](#):
 - Signal the [Commit Complete \(section 3.8.7.3\)](#) event on the transaction's Superior Enlistment's Transaction Manager Facet with the transaction's Superior Enlistment object.

- Signal the [Forget Transaction \(section 3.2.7.22\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the enlistment's Transaction object.

3.2.7.16 Enlistment Phase One Complete

The Enlistment Phase One Complete event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the enlistment's outcome for [Phase One \(section 1.3.1.2\)](#). This value MUST be set to one of the following values:
 - Committed
 - Aborted
 - In Doubt
 - Read Only
 - Prepared

If the Enlistment Phase One Complete event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- If the transaction's Doomed flag is set to true or the transaction state is [Aborting \(section 3.2.1.4.11\)](#), the Core Transaction Manager Facet (section 1.3.3.3.1) MUST ignore the signal.
- Otherwise:
 - Remove the enlistment from the transaction's Phase One Enlistment list.
 - If the transaction state is [Single Phase Commit \(section 3.2.1.4.9\)](#):
 - If the enlistment's Phase One outcome is Committed:
 - Set the Transaction's state to [Phase One Complete \(section 3.2.1.4.8\)](#).
 - Signal the [Phase One Complete \(section 3.4.7.13\)](#) event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The transaction's Superior Enlistment object.
 - The Committed outcome.
 - Signal the [Begin Commit \(section 3.2.7.2\)](#) event on the Core Transaction Manager Facet with the provided Transaction object.
 - Cease processing the event.
 - Otherwise, if the Enlistment's Phase One outcome is Read Only:
 - Signal the Phase One Complete event on the Transaction Manager Facet of the Transaction's Superior Enlistment with the following arguments:
 - The Transaction's Superior Enlistment object.
 - The Read Only outcome.

- Signal the [Forget Transaction \(section 3.2.7.22\)](#) event on the Core Transaction Manager Facet with the provided Transaction object.
- Otherwise, if the enlistment's Phase One outcome is [In Doubt \(section 3.2.1.4.12\)](#):
 - Set the Transaction's state to Phase One Complete.
 - Signal the Phase One Complete event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The transaction's Superior Enlistment object.
 - The In Doubt outcome.
 - Signal the [Begin In Doubt \(section 3.2.7.3\)](#) event on the Core Transaction Manager Facet with the provided Transaction object.
 - Cease processing the event.
- If the transaction state is Phase One or Single Phase Commit:
 - If the enlistment's Phase One outcome is Aborted:
 - Set the transaction's Doomed flag to true.
 - Signal the Phase One Complete event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The transaction's Superior Enlistment object.
 - The Aborted outcome.
 - Signal the [Notify Aborted \(section 3.2.7.23\)](#) event on the Core Transaction Manager Facet with the provided Transaction object.
 - Cease processing the event.
 - Otherwise, if the enlistment's Phase One outcome is Prepared.
 - Add the enlistment to the transaction's Phase Two Enlistment list.
- If both the transaction's Phase One Enlistment list and Phase One Enlistment list are now empty:
 - Signal the Phase One Complete event on the transaction's Superior Enlistment's Transaction Manager Facet with the provided Transaction object.

3.2.7.17 Enlistment Phase Zero Complete

The Enlistment Phase Zero Complete event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the enlistment's outcome for [Phase Zero \(section 1.3.1.1\)](#). This value MUST be set to one of the following values:
 - Completed
 - Aborted

If the Enlistment Phase Zero Complete event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Remove the enlistment from the transaction's Phase Zero Enlistments list.
- If the enlistment did not complete Phase Zero successfully:
 - Set the transaction's Doomed flag to true.
- If the transaction's Phase Zero Enlistments list is now empty:
 - Set the Transaction's state to [Phase Zero Complete \(section 3.2.1.4.4\)](#).
 - If the transaction's Doomed flag is set to true:
 - Signal the [Phase Zero Complete \(section 3.8.7.6\)](#) event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The transaction's Superior Enlistment object.
 - The Failure outcome.
 - Signal the [Notify Aborted \(section 3.2.7.23\)](#) event on the Core Transaction Manager Facet with the provided Transaction object.
 - Otherwise:
 - If the transaction's Root flag is true:
 - If the transaction's's Next Phase Zero Wave Enlistment list is not empty:
 - Set the Transaction's State to [Active \(section 3.2.1.4.2\)](#).
 - Signal the [Begin Phase Zero \(section 3.2.7.5\)](#) event on the Core Transaction Manager Facet with the provided enlistment's Transaction object.
 - Otherwise:
 - Signal the Phase Zero Complete event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The transaction's Superior Enlistment object.
 - The Success outcome.
 - Otherwise, if the transaction's Root flag is false:
 - If the transaction's Next Phase Zero WaveEnlistment list is not empty:
 - Set the Transaction's state to Active.
 - Signal the Phase Zero Complete event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The transaction's Superior Enlistment object.
 - The Success outcome.

3.2.7.18 Enlistment Rollback Complete

The Enlistment Rollback Complete event MUST be signaled with an Enlistment object.

If the Enlistment Rollback Complete event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Remove the enlistment from the transaction's Phase Two Enlistment list.
- If the transaction's Phase Two Enlistment list is now empty:
 - Signal the [Forget Transaction \(section 3.2.7.22\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the enlistment's Transaction object.

3.2.7.19 Enlistment Unilaterally Aborted

The Enlistment Unilaterally Aborted event MUST be signaled with the following arguments:

- An Enlistment object.

If the Enlistment Unilaterally Aborted event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- If the transaction state is [Active \(section 3.2.1.4.2\)](#), [Phase Zero \(section 3.2.1.4.3\)](#), [Phase Zero Complete \(section 3.2.1.4.4\)](#), [Voting \(section 3.2.1.4.5\)](#), [Voting Complete \(section 3.2.1.4.6\)](#) or [Phase One \(section 3.2.1.4.7\)](#):
 - Remove the provided Enlistment object from any of the following transaction lists in which it is present:
 - Next Phase Zero Wave Enlistment list
 - Phase Zero Enlistment list
 - Phase One Enlistment list
 - Phase One Voter Enlistment list
 - If the transaction state is Phase Zero (section 3.2.1.4.3):
 - Signal the [Phase Zero Complete \(section 3.8.7.6\)](#) event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The transaction's Superior Enlistment object
 - The Failure outcome
 - Otherwise, if the transaction state is Voting (section 3.2.1.4.5) or Core Transaction Manager Facet (section 1.3.3.3.1):
 - Signal the [Phase One Complete \(section 3.8.7.7\)](#) event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The transaction's Superior Enlistment object
 - The Aborted outcome
 - Otherwise:

- Signal the [Unilaterally Aborted \(section 3.8.7.11\)](#) event on the transaction's Superior Enlistment's Transaction Manager Facet with the transaction's Superior Enlistment object.
- Signal the [Notify Aborted \(section 3.2.7.23\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the provided Transaction object.
- Otherwise, ignore the event.

3.2.7.20 Enlistment Vote Complete

The Enlistment Vote Complete event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the enlistment's Vote. This value MUST be set to one of the following values:
 - Read Only
 - Prepared
 - Aborted

If the Enlistment Vote Complete event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- If the transaction's Doomed flag is set to true, the Core Transaction Manager Facet MUST cease processing the event.
- Otherwise:
 - If the enlistment's Vote outcome is Aborted:
 - Set the transaction's Doomed flag to true.
 - Remove the enlistment from the transaction's [Phase One \(section 1.3.1.2\)](#) Voter Enlistment list.
 - Signal the [Phase One Complete \(section 3.4.7.13\)](#) event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The transaction's Superior Enlistment object.
 - The Aborted outcome.
 - Signal the [Notify Aborted \(section 3.2.7.23\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the provided Transaction object.
 - Otherwise:
 - If the enlistment's Vote outcome is Read Only:
 - Remove the enlistment from the transaction's Phase One Voter Enlistment list.
 - Otherwise:
 - Move the enlistment from the transaction's Phase One Voter Enlistment list to the transaction's [Phase Two \(section 1.3.1.3\)](#) Voter Enlistment list.

- If the transaction's Phase One Voter Enlistment list is now empty:
 - If the transaction state is [Voting \(section 3.2.1.4.5\)](#):
 - Signal the [Voting Complete \(section 3.2.7.35\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the provided Transaction object.
 - Otherwise, if the transaction state is [Phase One \(section 3.2.1.4.7\)](#):
 - If both the transaction's Phase One Voter Enlistment list and Phase One Enlistment list are now empty:
 - Signal the Phase One Completed (section 3.4.7.13) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the provided Transaction object.

3.2.7.21 Export Transaction

The Export Transaction event MUST be signaled with the following arguments:

- A Transaction object.
- A Name object representing the remote subordinate transaction manager.

If the Export Transaction event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- If the transaction state is not [Active \(section 3.2.1.4.2\)](#), or [Phase Zero \(section 3.2.1.4.3\)](#), or [Phase Zero Complete \(section 3.2.1.4.4\)](#):
 - Signal the [Export Transaction Failure \(section 3.4.7.11\)](#) event on the Transaction Manager Communicating with an Application Facet with the following arguments:
 - The provided Transaction object.
 - The Too Late reason code.
- Otherwise, if the Durable Log is too full to accept a new enlistment:
 - Signal the Export Transaction Failure event on the Transaction Manager Communicating with an ApplicationFacet with the following arguments:
 - The provided Transaction object.
 - The Log Full reason code.
- Otherwise
 - Compute the number of Enlistment objects in the transaction's [Phase One](#) Enlistment list whose Transaction Manager Facet field is the Superior Transaction Manager.
 - If that number is equal to the Superior Transaction ManagerFacet's Max Enlistments Per Transaction field:
 - Signal the Export Transaction Failure event on the Transaction Manager Communicating with an Application Facet with the following arguments:
 - The provided Transaction object.

- The Too Many reason code.
- Otherwise:
 - Signal the [Propagate Transaction \(section 3.7.7.10\)](#) event on the Superior Transaction Manager Facet with the following arguments:
 - The provided Transaction object.
 - The provided Name object

3.2.7.22 Forget Transaction

The Forget Transaction event MUST be signaled with the following arguments:

- A Transaction object.

If the Forget Transaction event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Remove the provided Transaction object from the transaction table.
- If the transaction was added to the Core Transaction Manager Facet's (section 1.3.3.3.1) Durable Log:
 - Remove the transaction from the Durable Log.
- Set the Transaction's state to Ended.

3.2.7.23 Notify Aborted

The Notify Aborted event MUST be signaled with the following arguments:

- A Transaction object.

If the Notify Aborted event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Set the Transaction's state to [Aborting \(section 3.2.1.4.11\)](#).
- Move each Enlistment object in the Transaction's Next Phase Zero Enlistment list to the transaction's Phase Zero Enlistment list.
- For each Enlistment object in the Transaction's Phase Zero Enlistment list:
 - Signal the Phase Zero Aborted event (see sections [3.6.7.14](#) and [3.7.7.9](#)) on the enlistment's Transaction Manager Facet field with the Enlistment object.
- Move each Enlistment object in the transaction's [Phase One \(section 1.3.1.2\)](#) Voter Enlistment list to the transaction's [Phase Two \(section 1.3.1.3\)](#) Voter Enlistment list
- For each Enlistment object in the transaction's Phase Two Voter Enlistment list:
 - Signal the Begin Rollback event (sections [3.4.7.5](#), [3.6.7.5](#) and [3.7.7.4](#)) on the enlistment's Transaction Manager Facet field with the Enlistment object.
- Move the enlistment in the transaction's Phase One Voter Enlistment list to the transaction's Phase Two Voter Enlistment list

- If the transaction's Phase Two Enlistment list is not empty:
 - For each Enlistment object in the transaction's Phase Two Voter Enlistment list:
 - Signal the Begin Rollback event (sections 3.4.7.5, 3.6.7.5 and [3.7.7.4](#)) on the enlistment's Transaction Manager Facet field with the Enlistment object.
- Otherwise, if the transaction's Phase Two Enlistment list is empty:
 - Signal the [Forget Transaction \(section 3.2.7.22\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the provided Transaction object.

3.2.7.24 Notify Recovered Transaction Committed

The Notify Recovered Transaction Committed event MUST be signaled with the following arguments:

- A Transaction object.

If the Notify Recovered Transaction Committed event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Set the Transaction's state to [Failed to Notify \(section 3.2.1.4.13\)](#).
- If the transaction's Phase Two Enlistment list is not empty:
 - For each Enlistment object in the transaction's [Phase Two \(section 1.3.1.3\)](#) Voter Enlistment list:
 - Signal the [Begin Commit \(section 3.2.7.2\)](#) event on the enlistment's **Transaction Manager Facet** field with the Enlistment object.
- Otherwise:
 - Signal the [Forget Transaction \(section 3.2.7.22\)](#) event on the Core Transaction Manager Facet with the provided Transaction object.

3.2.7.25 Phase One Completed

The Phase One Completed event MUST be signaled by using the following arguments:

- A Transaction object.

If the Phase One Completed event is signaled, the Core Transaction Manager MUST perform the following actions:

- Set the state of the transaction to [Phase One Complete \(section 3.2.1.4.8\)](#).
- If both the transaction's Phase Two Enlistment list and the transaction's [Phase Two \(section 1.3.1.3\)](#) Voter Enlistment list are empty:
 - Signal the [Phase One Complete \(section 3.8.7.7\)](#) event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The transaction's Superior Enlistment object.
 - The Read-Only outcome.

- Signal the [Forget Transaction \(section 3.2.7.22\)](#) event on the [Core Transaction Manager Facet's \(section 1.3.3.3.1\)](#) with the provided Transaction object.
- If the [Single Phase Commit \(section 1.3.2.2\)](#) flag of the transaction is set to true:
 - Set the transaction state to [Failed to Notify \(section 3.2.1.4.13\)](#).
 - Save the transaction to the Durable Log of the Core Transaction Manager.
 - Signal the Phase One Complete event on the transaction's Superior Enlistment's Transaction Manager Facet using the following arguments:
 - The Superior Enlistment object of the transaction.
 - The Committed outcome.
 - Set the transaction state to Phase One Complete.
 - Signal the [Begin Commit \(section 3.2.7.2\)](#) event on the Core Transaction Manager Facet's with the provided Transaction object.
- Otherwise, if the Single Phase Commit flag of the transaction is set to false:
 - Set the transaction state to [In Doubt \(section 3.2.1.4.12\)](#).
 - Save the transaction to the Durable Log of the Core Transaction Manager.
 - Set the transaction state to Phase One Complete.
 - Signal the Phase One Complete event on the transaction's Superior Enlistment's Transaction Manager Facet using the following arguments:
 - The Superior Enlistment object of the transaction.
 - The Prepared outcome.

3.2.7.26 Propagate Transaction Failure

The Propagate Transaction Failure event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:
 - No Mem
 - Log Full
 - Duplicate

If the Propagate Transaction Failure event is signaled, the [Core Transaction Manager Facet's \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Signal the [Export Transaction Failure \(section 3.4.7.11\)](#) event on the Transaction Manager Communicating with an Application Facet with the following arguments:
 - The provided Transaction object.
 - The provided reason code.

3.2.7.27 Propagate Transaction Success

The Propagate Transaction Success event MUST be signaled with the following arguments:

- An Enlistment object.

If the Propagate Transaction Success event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- If the enlistment's transaction is not [Active \(section 3.2.1.4.2\)](#), or [Phase Zero \(section 3.2.1.4.3\)](#), or [Phase Zero Complete \(section 3.2.1.4.4\)](#):
 - Signal the [Export Transaction Failure \(section 3.4.7.11\)](#) event on the Transaction Manager Communicating with an Application Facet with the following arguments:
 - The provided Transaction object.
 - The Too Late reason code.
 - Set the state of the connection referenced by the Enlistment object to [Ended \(section 3.2.1.4.14\)](#).
- Otherwise:
 - Add the Enlistment object to the transaction's Phase One Enlistment list.
 - Signal the Export Transaction Success (section 3.4.7.11) event on the Transaction Manager Communicating with an Application Facet with the enlistment's Transaction object.

3.2.7.28 Register Phase Zero Failure

The Register Phase Zero Failure event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:
 - Too Late
 - Tx Not Found

If the Register Phase Zero Failure event is signaled, the Core Transaction Manager MUST perform the following actions:

- For each Enlistment object in the transaction's Next Phase Zero Wave Enlistment list:
 - Signal the [Create Phase Zero Enlistment Failure](#) event (see sections [3.6.7.7](#) and [3.7.7.5](#)) on the Enlistment object's Transaction Manager Facet with the following arguments:
 - The Enlistment object.
 - The provided reason code.
 - Remove the Enlistment object from the list.

3.2.7.29 Register Phase Zero Success

The Register Phase Zero Success event MUST be signaled with the following arguments:

- An Enlistment object.

If the Register Phase Zero Success event is signaled, the Core Transaction Manager MUST perform the following actions:

- For each Enlistment object in the transaction's Next Phase Zero Wave Enlistment list:
 - Signal the [Create Phase Zero Enlistment Success](#) event (see sections [3.6.7.8](#) and [3.7.7.6](#)) on the Enlistment object's Transaction Manager Facet with the Enlistment object.
- Set the Phase Zero Registered flag of the Transaction object referenced by the Enlistment to true.

3.2.7.30 Resolve Transaction

The Resolve Transaction event MUST be signaled with the following arguments:

- A Transaction object.
- A value indicating the desired resolve transaction outcome. This value MUST be set to one of the following values:
 - Committed
 - Aborted
 - Forgotten

If the Resolve Transaction event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- If the provided resolved transaction outcome is Committed or Aborted:
 - If the transaction state is not [In Doubt \(section 3.2.1.4.12\)](#):
 - Signal the [Resolve Transaction Complete \(section 3.4.7.16\)](#) event on the Transaction Manager Communicating with an Application Facet, with the following arguments:
 - The provided Transaction object.
 - The Not Prepared result.
 - Otherwise:
 - If the provided resolve transaction outcome is Committed:
 - Signal the [Notify Recovered Transaction Committed \(section 3.2.7.24\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the provided Transaction object.
 - Signal the Resolve Transaction Complete (section 3.4.7.16) event on the Transaction Manager Communicating with an Application Facet with the following arguments:
 - The provided Transaction object.
 - The Committed result.

- Otherwise, if the provided resolve transaction outcome is Aborted:
 - Signal the [Notify Aborted \(section 3.2.7.23\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the provided Transaction object.
 - Signal the Resolve Transaction Complete (section 3.4.7.16) event on the Transaction Manager Communicating with an Application Facet with the following arguments:
 - The provided Transaction object.
 - The Aborted result.
- Otherwise:
 - If the transaction state is not [Failed to Notify \(section 3.2.1.4.13\)](#):
 - Signal the Resolve Transaction Complete (section 3.4.7.16) event on the Transaction Manager Communicating with an Application Facet with the following arguments:
 - The provided Transaction object.
 - The Not Committed result.
 - Otherwise:
 - Set each Enlistment object in the transaction's Phase Two Enlistment list to [Ended \(section 3.2.1.4.14\)](#).
 - Signal the [Forget Transaction \(section 3.2.7.22\)](#) event on the Core Transaction Manager Facet's (section 1.3.3.3.1) own Forget Transaction (section 3.2.7.22) event Facet with the Transaction object.
 - Signal the Resolve Transaction Complete (section 3.4.7.16) event on the Transaction Manager Communicating with an Application Facet, with the following arguments:
 - The provided Transaction object.
 - The Forgotten result.

3.2.7.31 Set Transaction Attributes

The Set Transaction Attributes event MUST be signaled with the following arguments:

- A Transaction object.
- A value indicating the transaction's Isolation Level. The value MUST be one of the isolation level values specified in section [2.2.6.9](#).
- A value indicating the transaction's Isolation Flags. The value MUST be one of the valid isolation flag values specified in section [2.2.6.8](#)
- A string indicating the an implementation-specific description of the transaction.

If the Set Transaction Attributes event is signaled, the Core Transaction Manager MUST perform the following actions:

- If the Transaction's state is not Active:

- Signal the [Set Transaction Attributes Failure \(section 3.4.7.18\)](#) event on the Transaction Manager Communicating with an Application Facet with the Transaction object.
- Otherwise
 - If the Transaction object's Attributes Set flag is set to false:
 - Set the Transaction object's **Isolation Level** field with the Isolation Level argument
 - Set the Transaction object's **Isolation Flags** field with the Isolation Flags argument
 - Set the Transaction object's **Description** field with the Description Argument
 - Set the Transaction object's Attributes Set flag to true.
 - Signal the [Set Transaction Attributes Success \(section 3.4.7.19\)](#) event on the Transaction Manager Communicating with an Application Facet with the Transaction object.

3.2.7.32 Set Transaction Timeout

The Set Transaction Timeout event MUST be signaled with the following arguments:

- A Transaction object.
- A timespan.

If the Set Transaction Timeout event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- If the transaction's Transaction Timeout timer has already been canceled:
 - Signal the [Set Transaction Timeout Failure \(section 3.4.7.20\)](#) event on the Transaction Manager Communicating with an Application Facet with the provided Transaction object.
- Otherwise:
 - Set the transaction's **Timeout** field to the provided value.
 - Update the timer's timeout value to the provided timespan value.
 - Signal the [Set Transaction Timeout Success \(section 3.4.7.21\)](#) event on the Transaction Manager Communicating with an Application Facet with the Transaction object.

3.2.7.33 Request Transaction Outcome

The Request Transaction Outcome MUST be signaled with the following arguments:

- An Enlistment object.

If the Request Transaction Outcome event is signaled, the Core Transaction Manager MUST perform the following actions:

- If the state of the Transaction object referenced by the provided Enlistment object is [Committing \(section 3.2.1.4.10\)](#) or [Failed to Notify \(section 3.2.1.4.13\)](#):
 - Signal the [Begin Commit \(section 3.2.7.2\)](#) event on the provided Enlistment object's Transaction ManagerFacet with the provided Enlistment object.

- Otherwise, if the provided enlistment's transaction state is [Aborting \(section 3.2.1.4.11\)](#) or [Ended \(section 3.2.1.4.14\)](#):
 - Signal the [Begin Rollback \(section 3.4.7.5\)](#) event on the provided Enlistment object's Transaction Manager Facet with the provided Enlistment object.
- Otherwise ignore the event.

3.2.7.34 Unenlist Phase Zero Enlistment

The Unenlist Phase Zero Enlistment event MUST be signaled with the following arguments:

- An enlistment object.

If the Unenlist Phase Zero Enlistment event is signaled, the core transaction manager MUST perform the following actions:

- If the provided Enlistment object is a member of the transaction's next Phase Zero Wave enlistment list:
 - Remove the Enlistment object from the list.
- Otherwise, if the provided Enlistment object is a member of the transaction's Phase Zero enlistment list:
 - Remove the Enlistment object from the list.

3.2.7.35 Voting Complete

The Voting Complete event MUST be signaled by using the following arguments:

- A Transaction object.

If the Voting Complete event is signaled, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST perform the following actions:

- Set the transaction state to Voting Complete.
- If the Phase One Enlistment list of the transaction is empty:
 - If the [Phase Two \(section 1.3.1.3\)](#) Voter Enlistment list of the transaction is empty:
 - Signal the [Phase One Complete \(section 3.4.7.13\)](#) event on the transaction's Superior Enlistment's Transaction Manager Facet using the following arguments:
 - The Superior Enlistment that is referenced by the provided Transaction object.
 - The Read Only outcome.
 - Set the Transaction's State to [Ended \(section 3.2.1.4.14\)](#).
 - Otherwise:
 - If the transaction's [Single Phase Commit \(section 3.2.1.4.9\)](#) flag is set to true:
 - Signal the Phase One Complete event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:

- The Superior Enlistment referenced by the provided Transaction object.
- The Committed outcome.
- Set the Transaction's State to Phase One Complete.
- Signal the [Begin Commit \(section 3.2.7.2\)](#) event on the Core Transaction Manager with the provided Transaction object.
- Otherwise:
 - Set the Transaction's state to Phase One Complete.
 - Signal the Phase One Complete event on the transaction's Superior Enlistment's Transaction Manager Facet with the following arguments:
 - The Superior Enlistment referenced by the provided Transaction object.
 - The Prepared outcome.
- Otherwise, if the transaction's Single Phase Commit flag is set to true and the transaction's Phase One Enlistment list contains one element:
 - Set the Transaction's state to Single Phase Commit.
 - Signal the [Begin Phase One \(section 3.2.7.4\)](#) event on the Transaction ManagerFacet of the Phase One Enlistment object with the following arguments:
 - The Enlistment object.
 - The Single Phase Commit flag set to true.
- Otherwise:
 - Set the Transaction's State to [Phase One](#).
 - For each Enlistment object in the transaction'sPhase One Enlistment list:
 - Signal the Begin Phase One (section 3.2.7.4) event (see also the Resource Manager and Superior Transaction Manager **Begin Phase One** events in sections [3.6.7.3](#) and [3.7.7.2](#), respectively) on the enlistment's **Transaction Manager Facet** field with the following argument:
 - The Enlistment object
 - The Single Phase Commit (section 3.2.1.4.9) flag set to false

3.3 Application Details

3.3.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with the behavior that is described in this document.

An application **MUST** maintain all the data elements that are specified in section [3.1.1](#).

An application MUST also maintain the following data elements:

- **Transaction Manager Name:** A Name object that identifies the transaction manager that is associated with the application.

An application MUST provide the states that are defined in the following sections for its supported connection types. Section [2.2.8](#) defines the connection types that an application MUST provide for each supported protocol version.

3.3.1.1 CONNTYPE_TXUSER_BEGINNER Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Begin Response
- Processing Transaction
- Awaiting Commit Response
- Awaiting Abort Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_BEGINNER initiator states.

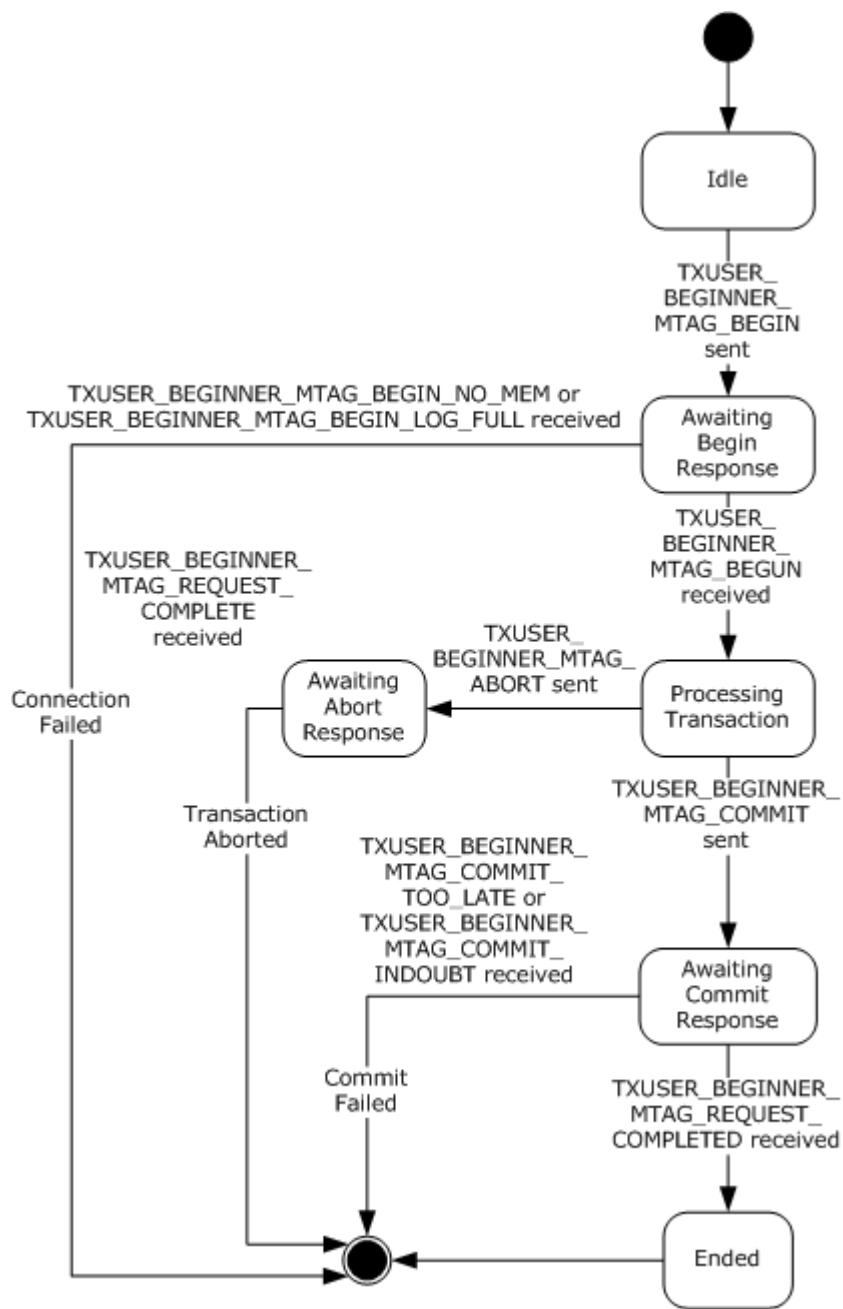


Figure 14: CONNTYPE_TXUSER_BEGINNER initiator states

3.3.1.1.1 Idle

This is the initial state. The following event is processed in this state:

- [Beginning a Transaction Using CONNTYPE_TXUSER_BEGINNER \(section 3.3.4.1.2\).](#)

3.3.1.1.2 Awaiting Begin Response

The following events are processed in this state:

- [Receiving a TXUSER BEGINNER MTAG BEGUN Message \(section 3.3.5.1.1.1\).](#)
- [Receiving a TXUSER BEGINNER MTAG BEGIN NO MEM or TXUSER BEGINNER MTAG BEGIN LOG FULL Message \(section 3.3.5.1.1.2\).](#)

3.3.1.1.3 Processing Transaction

The following events are processed in the Processing Transaction state:

- [Initiating Transaction Commit \(section 3.3.4.7\).](#)
- [Initiating Transaction Rollback \(section 3.3.4.8\).](#)

3.3.1.1.4 Awaiting Commit Response

The following events are processed in this state:

- [Receiving a TXUSER BEGINNER MTAG REQUEST COMPLETED Message \(section 3.3.5.1.1.3\).](#)
- [Receiving a TXUSER BEGINNER MTAG COMMIT TOO LATE Message \(section 3.3.5.1.1.4\).](#)
- [Receiving a TXUSER BEGINNER MTAG COMMIT INDOUBT Message \(section 3.3.5.1.1.5\).](#)

3.3.1.1.5 Awaiting Abort Response

The following event is processed in this state:

- [Receiving a TXUSER BEGINNER MTAG REQUEST COMPLETED Message \(section 3.3.5.1.1.3\).](#)

3.3.1.1.6 Ended

This is the final state.

3.3.1.2 CONNTYPE_TXUSER_BEGIN2 Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Begin Response
- Processing Transaction
- Awaiting Set Timeout Response
- Awaiting Commit Response
- Awaiting Abort Response

The following figure shows the relationship between the CONNTYPE_TXUSER_BEGIN2 initiator states.

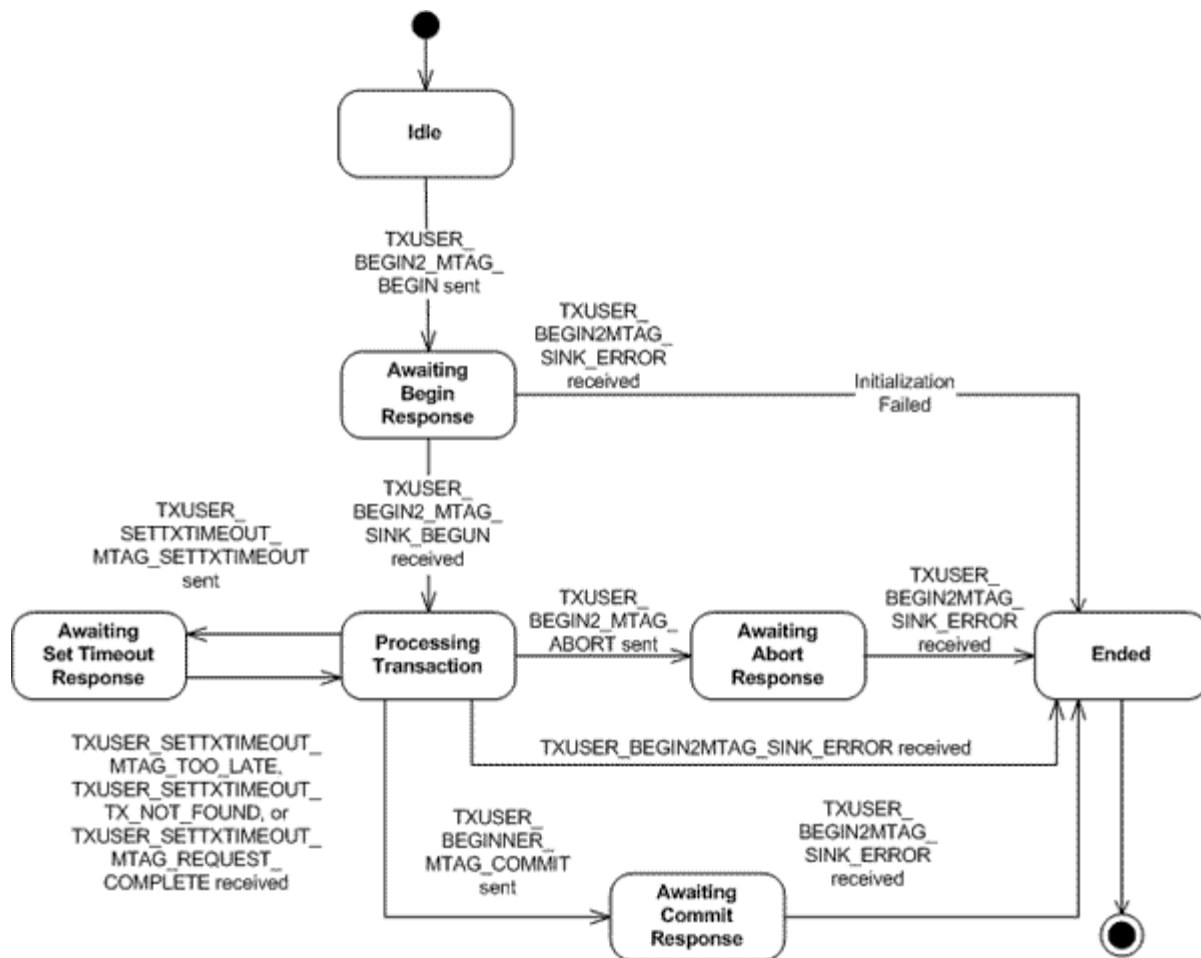


Figure 15: CONNTYPE_TXUSER_BEGIN2 initiator states

3.3.1.2.1 Idle

This is the initial state. The following event is processed in this state:

- [Beginning a Transaction Using CONNTYPE_TXUSER_BEGIN2 \(section 3.3.4.1.1\).](#)

3.3.1.2.2 Awaiting Begin Response

The following events are processed in this state:

- [Receiving a TXUSER_BEGIN2_MTAG_SINK_BEGUN Message \(section 3.3.5.1.2.1\).](#)
- [Receiving a TXUSER_BEGIN2_MTAG_SINK_ERROR Message \(section 3.3.5.1.2.4\).](#)

3.3.1.2.3 Processing Transaction

The following events are processed in the Processing Transaction state:

- [Changing a Transaction Time-out Using CONNTYPE_TXUSER_SETTXTIMEOUT2 \(section 3.3.4.2.2\).](#)

- [Commit a Transaction Using CONNTYPE_TXUSER_BEGIN2 \(section 3.3.4.7.1\).](#)
- [Abort a Transaction Using CONNTYPE_TXUSER_BEGIN2 \(section 3.3.4.8.1\).](#)

3.3.1.2.4 Awaiting Set Timeout Response

The following events are processed in this state:

- [Receiving a TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE Message \(section 3.3.5.1.2.2\).](#)
- [Receiving a TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE Message \(section 3.3.5.1.2.3\).](#)
- [Receiving a TXUSER_SETTXTIMEOUT_TX_NOT_FOUND Message \(section 3.3.5.3.4.1\).](#)

3.3.1.2.5 Awaiting Commit Response

The following event is processed in the Awaiting Commit Response state:

- [Receiving a TXUSER_BEGIN2_MTAG_SINK_ERROR Message \(section 3.3.5.1.2.4\).](#)

3.3.1.2.6 Awaiting Abort Response

The following event is processed in this state:

- [Receiving a TXUSER_BEGIN2_MTAG_SINK_ERROR Message \(section 3.3.5.1.2.4\).](#)

3.3.1.2.7 Ended

This is the final state.

3.3.1.3 CONNTYPE_TXUSER_PROMOTE Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Promote Response
- Processing Transaction
- Awaiting Timeout Response
- Awaiting Commit Response
- Awaiting Abort Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_PROMOTE initiator states.

- [Receiving a TXUSER BEGIN2 MTAG SINK BEGUN Message \(section 3.3.5.1.2.1\).](#)
- [Receiving a TXUSER BEGIN2 MTAG SINK ERROR Message \(section 3.3.5.1.2.4\).](#)

3.3.1.3.3 Processing Transaction

The following events are processed in the Processing Transaction state:

- [Commit a Transaction Using CONNTYPE TXUSER PROMOTE \(section 3.3.4.7.3\).](#)
- [Roll Back a Transaction Using CONNTYPE TXUSER PROMOTE \(section 3.3.4.8.5\).](#)
- [Changing a Transaction Time-out Using CONNTYPE TXUSER SETTXTIMEOUT2 \(section 3.3.4.2.2\).](#)

3.3.1.3.4 Awaiting Timeout Response

The following events are processed in this state:

- [Receiving a TXUSER SETTXTIMEOUT MTAG REQUEST COMPLETE Message \(section 3.3.5.1.2.2\).](#)
- [Receiving a TXUSER SETTXTIMEOUT MTAG TOO LATE or TXUSER SETTXTIMEOUT MTAG TX NOT FOUND Message \(section 3.3.5.3.3.2\).](#)

3.3.1.3.5 Awaiting Commit Response

The following event is processed in this state:

- [Receiving a TXUSER BEGIN2 MTAG SINK ERROR Message \(section 3.3.5.1.2.4\).](#)

3.3.1.3.6 Awaiting Abort Response

The following event is processed in this state:

- [Receiving a TXUSER BEGIN2 MTAG SINK ERROR Message \(section 3.3.5.1.2.4\).](#)

3.3.1.3.7 Ended

This is the final state.

3.3.1.4 CONNTYPE_TXUSER_ASSOCIATE Initiator States

The application MUST act as an initiator for the [CONNTYPE TXUSER ASSOCIATE \(section 2.2.8.2.1.1\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Associate Response
- Active
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_ASSOCIATE initiator states.

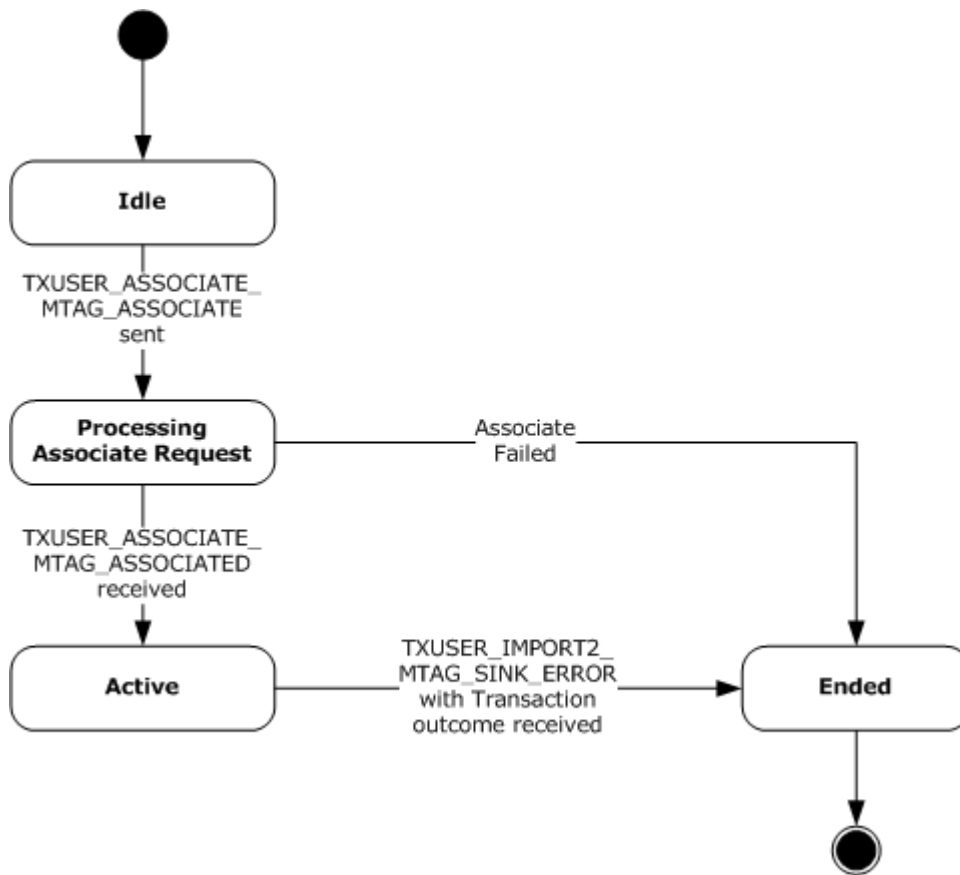


Figure 17: CONNTYPE_TXUSER_ENLISTMENT initiator states

3.3.1.4.1 Idle

This is the initial state. The following event is processed in this state:

- [Pulling a Transaction \(section 3.3.4.11\)](#).

3.3.1.4.2 Awaiting Associate Response

The following events are processed in this state:

- [Receiving a TXUSER_ASSOCIATE_MTAG_ASSOCIATED Message \(section 3.3.5.2.1.1.1\)](#).
- [Receiving Other TXUSER_ASSOCIATE_MTAG Messages \(section 3.3.5.2.1.1.2\)](#).

3.3.1.4.3 Active

The following events are processed in this state:

- Receiving a [TXUSER_IMPORT2_MTAG_SINK_ERROR \(section 3.3.5.2.1.1.3\)](#) Message.

3.3.1.4.4 Ended

This is the final state.

3.3.1.5 CONNTYPE_TXUSER_EXTENDWHEREABOUTS Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS \(section 2.2.8.2.2.1\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Get Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_EXTENDWHEREABOUTS initiator states.

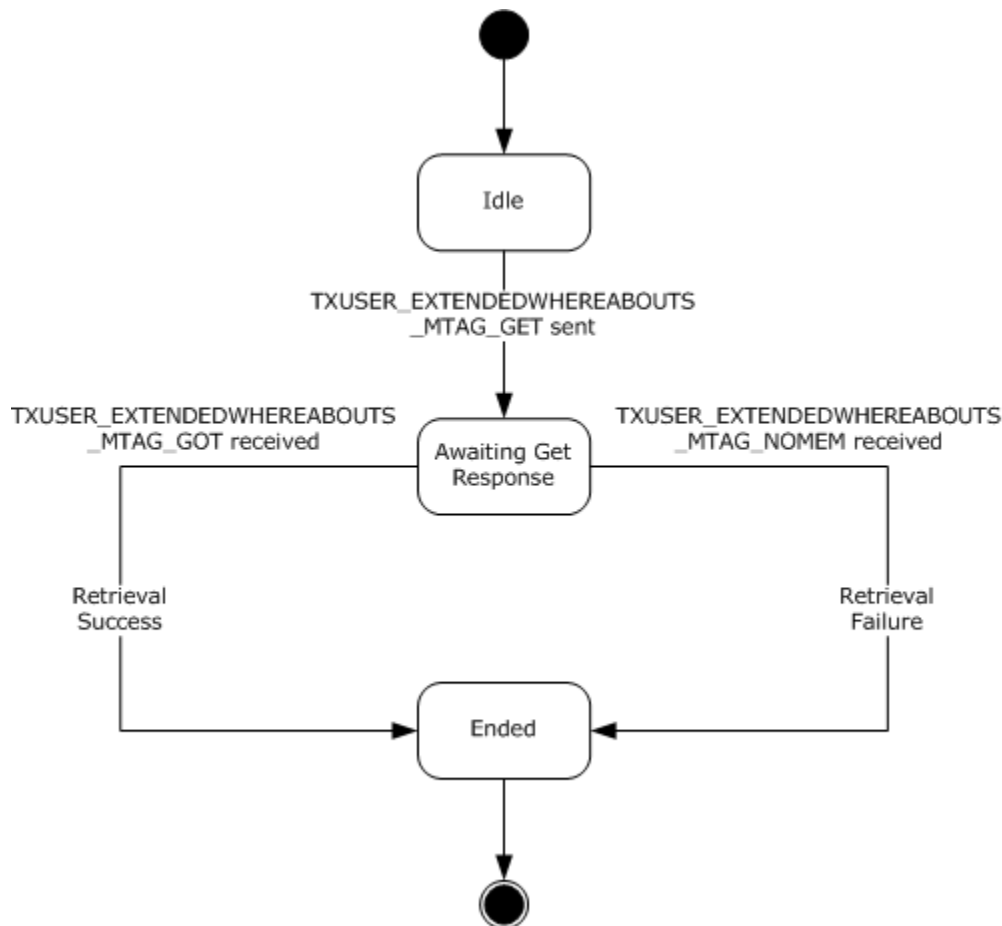


Figure 18: CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS initiator states

3.3.1.5.1 Idle

This is the initial state. The following event is processed in this state:

- [Obtaining Extended Whereabouts Using CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS \(section 3.3.4.9\)](#).

3.3.1.5.2 Awaiting Get Response

The following events are processed in this state:

- [Receiving a TXUSER_EXTENDEDWHEREABOUTS_MTAG_GOT Message \(section 3.3.5.2.2.1.1\).](#)
- [Receiving a TXUSER_EXTENDEDWHEREABOUTS_MTAG_NOMEM Message \(section 3.3.5.2.2.1.2\).](#)

3.3.1.5.3 Ended

This is the final state.

3.3.1.6 CONNTYPE_TXUSER_IMPORT Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_IMPORT \(section 2.2.8.2.2.3\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Import Response
- Transaction Import Successful
- Awaiting Abort Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_IMPORT initiator states:

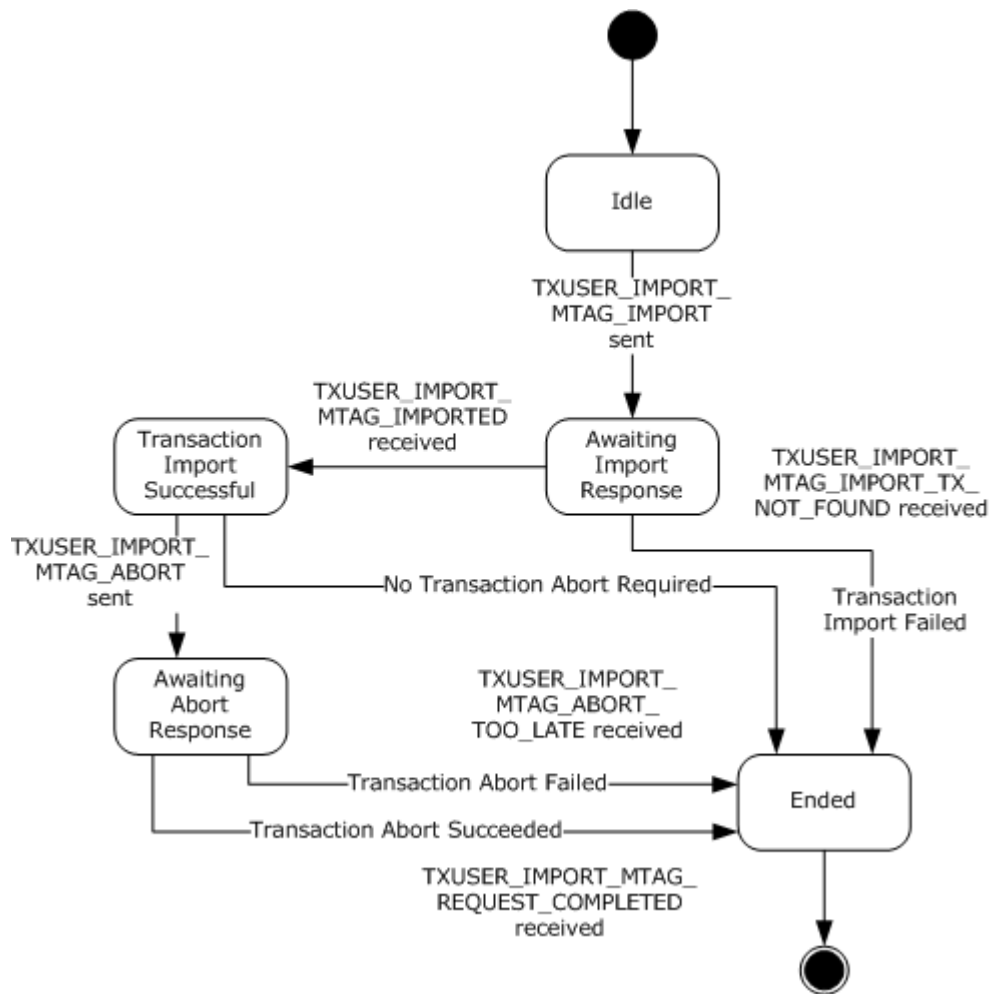


Figure 19: CONNTYPE_TXUSER_IMPORT initiator states

3.3.1.6.1 Idle

This is the initial state. The following event is processed in this state:

- [Importing a Transaction Using CONNTYPE_TXUSER_IMPORT \(section 3.3.4.5.1\).](#)

3.3.1.6.2 Awaiting Import Response

The following events are processed in this state:

- [Receiving a TXUSER_IMPORT_MTAG_IMPORTED Message \(section 3.3.5.2.2.3.1\).](#)
- [Receiving a TXUSER_IMPORT_MTAG_IMPORT_TX_NOT_FOUND Message \(section 3.3.5.2.2.3.2\).](#)

3.3.1.6.3 Transaction Import Successful

The following event is processed in this state:

- [Abort a Transaction Using CONNTYPE_TXUSER_IMPORT \(section 3.3.4.8.3\).](#)

3.3.1.6.4 Awaiting Abort Response

The following events are processed in this state:

- [Receiving a TXUSER_IMPORT_MTAG_ABORT_TOO_LATE Message \(section 3.3.5.2.3.2\).](#)
- [Receiving a TXUSER_IMPORT_MTAG_REQUEST_COMPLETED Message \(section 3.3.5.2.3.3\).](#)

3.3.1.6.5 Ended

This is the final state.

3.3.1.7 CONNTYPE_TXUSER_IMPORT2 Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.2.4\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Import Response
- Transaction Import Successful
- Awaiting Abort Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_IMPORT2 initiator states.

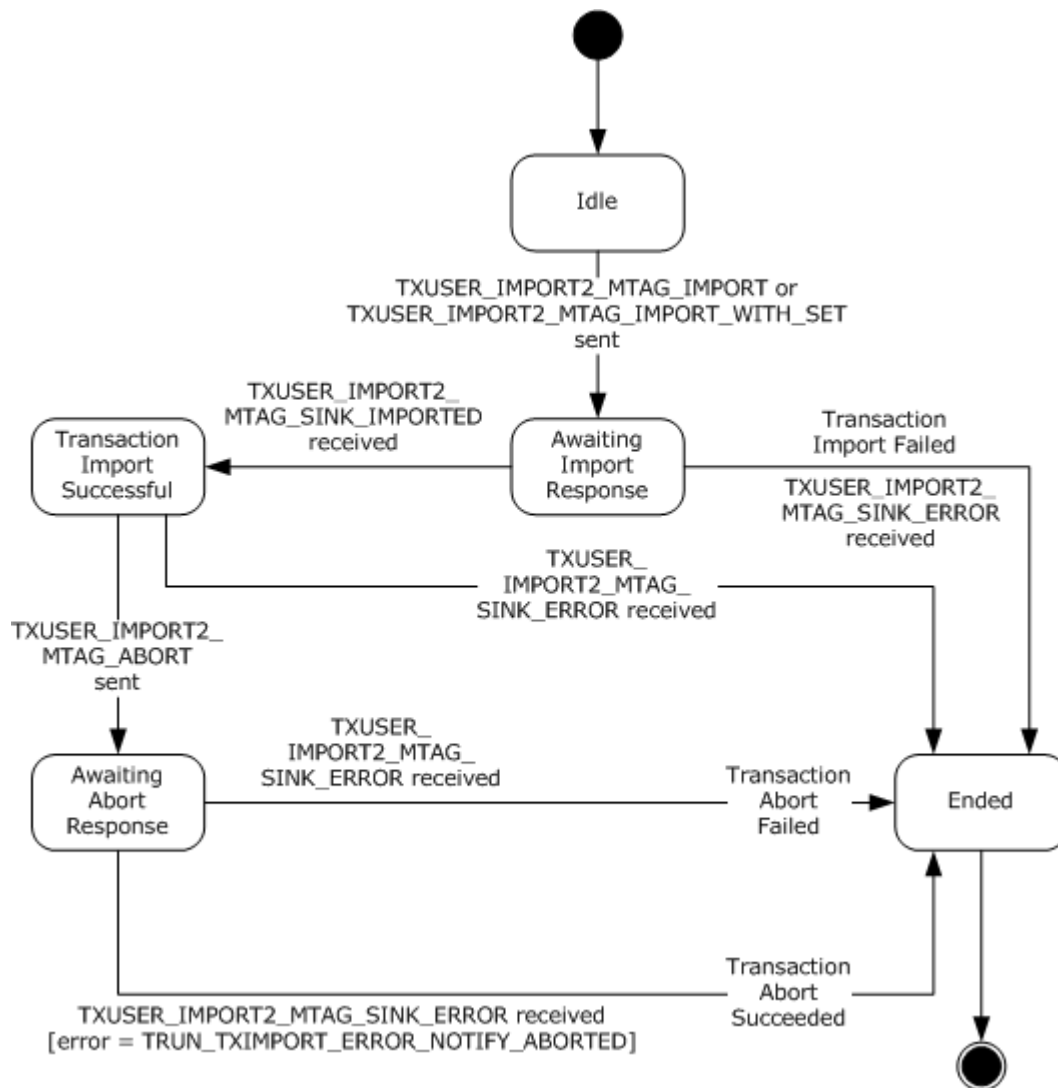


Figure 20: CONNTYPE_TXUSER_IMPORT2 initiator states

3.3.1.7.1 Idle

This is the initial state. The following events are processed in this state:

- [Importing a Transaction Using CONNTYPE_TXUSER_IMPORT2 \(section 3.3.4.5.2\).](#)
- [Importing a Transaction with Additional Transaction Attributes \(section 3.3.4.6\).](#)

3.3.1.7.2 Awaiting Import Response

The following events are processed in this state:

- [Receiving a TXUSER_IMPORT2_MTAG_IMPORTED Message \(section 3.3.5.2.2.4.1\).](#)
- [Receiving a TXUSER_IMPORT2_MTAG_SINK_ERROR Message \(section 3.3.5.2.2.4.2\).](#)

3.3.1.7.3 Transaction Import Successful

The following events are processed in this state:

- [Receiving a TXUSER_IMPORT2_MTAG_SINK_ERROR Message \(section 3.3.5.2.2.4.2\).](#)
- [Abort a Transaction Using CONNTYPE_TXUSER_IMPORT2 \(section 3.3.4.8.4\).](#)

3.3.1.7.4 Awaiting Abort Response

The following event is processed in this state:

- [Receiving a TXUSER_IMPORT2_MTAG_SINK_ERROR Message \(section 3.3.5.2.2.4.2\).](#)

3.3.1.7.5 Ended

This is the final state.

3.3.1.8 CONNTYPE_TXUSER_EXPORT Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_EXPORT \(section 2.2.8.2.2.2\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Create Response
- Connection Active
- Awaiting Export Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_EXPORT initiator states.

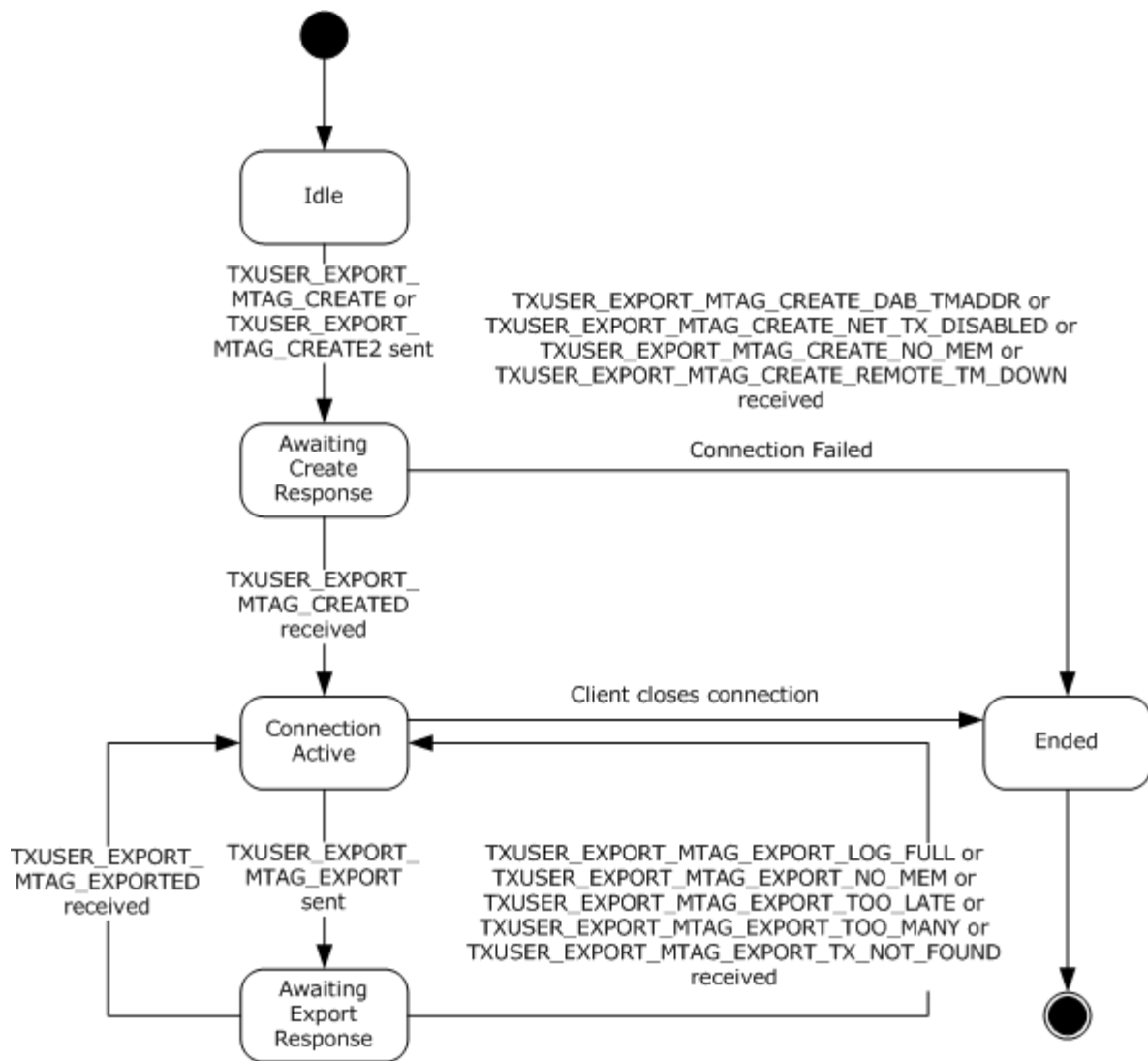


Figure 21: CONNTYPE_TXUSER_EXPORT initiator states

3.3.1.8.1 Idle

This is the initial state. The following event is processed in this state:

- [Creating an Export Connection \(section 3.3.4.3\).](#)

3.3.1.8.2 Awaiting Create Response

The following events are processed in this state:

- [Receiving a TXUSER_EXPORT_MTAG_CREATED Message \(section 3.3.5.2.2.2.1\).](#)
- [Receiving a TXUSER_EXPORT_MTAG_CREATE_BAD_TMADDR or TXUSER_EXPORT_MTAG_CREATE_NET_TX_DISABLED Message \(section 3.3.5.2.2.2.2\).](#)

3.3.1.8.3 Connection Active

The following event is processed in this state:

- [Push a Transaction Using an Existing Export Connection \(section 3.3.4.12\).](#)

3.3.1.8.4 Awaiting Export Response

The following events are processed in this state:

- [Receiving a TXUSER_EXPORT_MTAG_EXPORTED Message \(section 3.3.5.2.2.3\).](#)
- [Receiving a TXUSER_EXPORT_MTAG_EXPORT_LOG_FULL, TXUSER_EXPORT_MTAG_EXPORT_NO_MEM, TXUSER_EXPORT_MTAG_EXPORT_TOO_LATE, TXUSER_EXPORT_MTAG_EXPORT_TOO_MANY, or TXUSER_EXPORT_MTAG_EXPORT_NOT_FOUND Message \(section 3.3.5.2.2.4\).](#)

3.3.1.8.5 Ended

This is the final state.

3.3.1.9 CONNTYPE_TXUSER_GETTXDETAILS Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_GETTXDETAILS \(section 2.2.8.3.1\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_GETTXDETAILS initiator states.

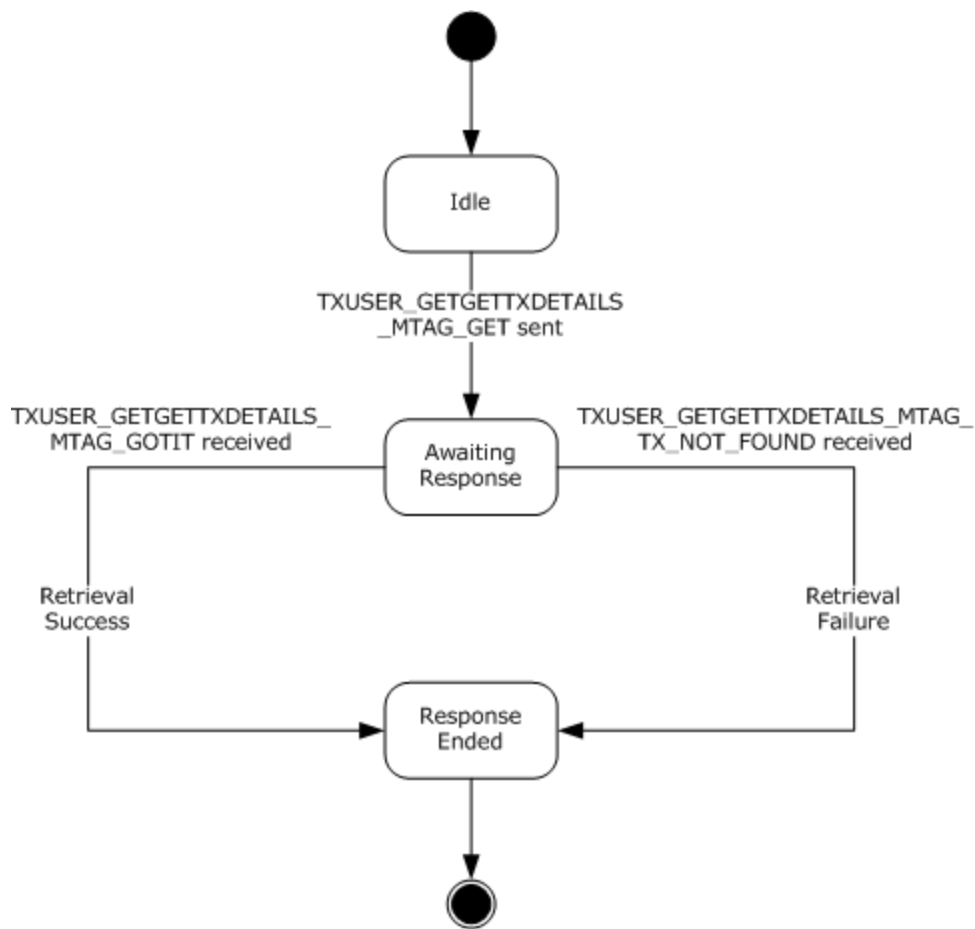


Figure 22: CONNTYPE_TXUSER_GETTXDETAILS initiator states

3.3.1.9.1 Idle

This is the initial state. The following event is processed in this state:

- [Obtaining the Details for a Transaction \(section 3.3.4.10.1\).](#)

3.3.1.9.2 Awaiting Response

The following events are processed in this state:

- [Receiving a TXUSER_GETTXDETAILS_MTAG_GOTIT Message \(section 3.3.5.3.1.1\).](#)
- [Receiving a TXUSER_GETTXDETAILS_MTAG_TX_NOT_FOUND Message \(section 3.3.5.3.1.2\).](#)

3.3.1.9.3 Ended

This is the final state.

3.3.1.10 CONNTYPE_TXUSER_RESOLVE Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_RESOLVE \(section 2.2.8.3.2\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Abort Response
- Awaiting Forget Response
- Awaiting Commit Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_RESOLVE initiator states.

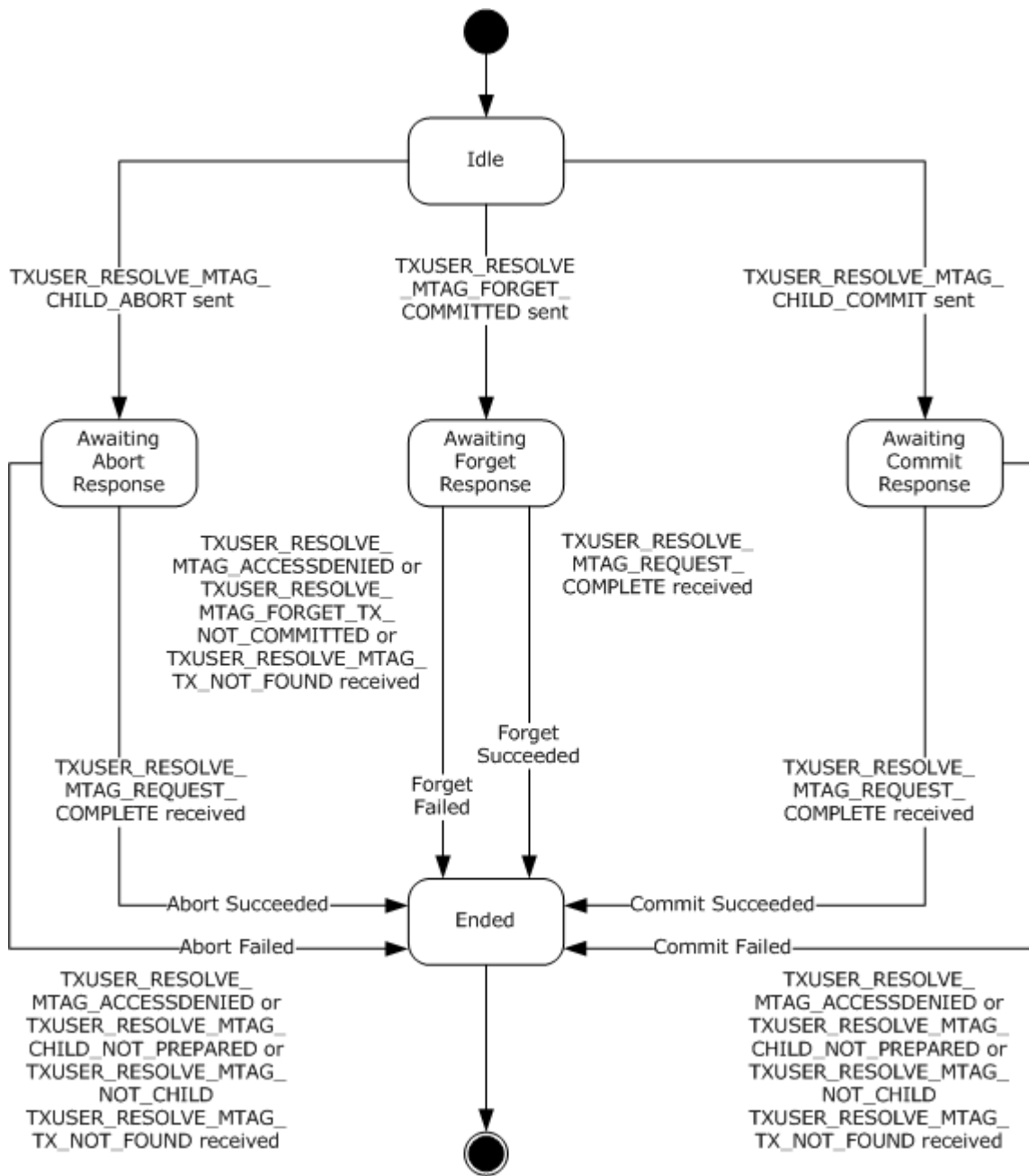


Figure 23: CONNTYPE_TXUSER_RESOLVE initiator states

3.3.1.10.1 Idle

This is the initial state. The following event is processed in this state:

- [Resolving a Transaction \(section 3.3.4.13\).](#)

3.3.1.10.2 Awaiting Abort Response

The following events are processed in this state:

- [Receiving a TXUSER_RESOLVE_MTAG_REQUEST_COMPLETE Message \(section 3.3.5.3.2.1\).](#)
- [Receiving a TXUSER_RESOLVE_MTAG_CHILD_NOT_PREPARED Message \(section 3.3.5.3.2.3\).](#)
- [Receiving a TXUSER_RESOLVE_MTAG_ACCESSDENIED or TXUSER_RESOLVE_MTAG_TX_NOT_FOUND Message \(section 3.3.5.3.2.2\).](#)

3.3.1.10.3 Awaiting Forget Response

The following events are processed in this state:

- [Receiving a TXUSER_RESOLVE_MTAG_REQUEST_COMPLETE Message \(section 3.3.5.3.2.1\).](#)
- [Receiving a TXUSER_RESOLVE_MTAG_FORGET_TX_NOT_COMMITTED Message \(section 3.3.5.3.2.4\).](#)
- [Receiving a TXUSER_RESOLVE_MTAG_ACCESSDENIED or TXUSER_RESOLVE_MTAG_TX_NOT_FOUND Message \(section 3.3.5.3.2.2\).](#)

3.3.1.10.4 Awaiting Commit Response

The following events are processed in this state:

- [Receiving a TXUSER_RESOLVE_MTAG_REQUEST_COMPLETE Message \(section 3.3.5.3.2.1\).](#)
- [Receiving a TXUSER_RESOLVE_MTAG_CHILD_NOT_PREPARED Message \(section 3.3.5.3.2.3\).](#)
- [Receiving a TXUSER_RESOLVE_MTAG_ACCESSDENIED or TXUSER_RESOLVE_MTAG_TX_NOT_FOUND Message \(section 3.3.5.3.2.2\).](#)

3.3.1.10.5 Ended

This is the final state.

3.3.1.11 CONNTYPE_TXUSER_SETTXTIMEOUT Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_SETTXTIMEOUT \(section 2.2.8.3.3\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Set Timeout Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_SETTXTIMEOUT initiator states.

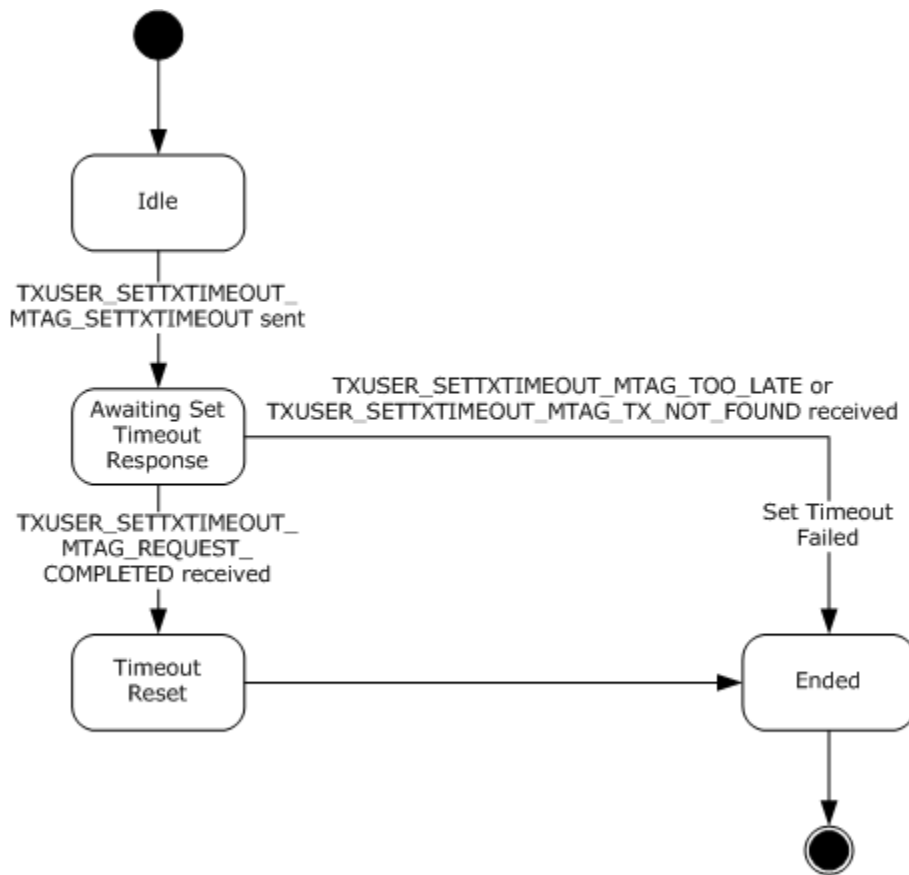


Figure 24: CONNTYPE_TXUSER_SETTXXTIMEOUT initiator states

3.3.1.11.1 Idle

This is the initial state. The following event is processed in this state:

- [Changing a Transaction Time-out Using CONNTYPE_TXUSER_SETTXXTIMEOUT \(section 3.3.4.2.1\).](#)

3.3.1.11.2 Awaiting Set Timeout Response

The following events are processed in the Awaiting Set Timeout Response state:

- [Receiving a TXUSER_SETTXXTIMEOUT_MTAG_REQUEST_COMPLETE Message \(section 3.3.5.1.2.2\).](#)
- [Receiving a TXUSER_SETTXXTIMEOUT_MTAG_TOO_LATE or TXUSER_SETTXXTIMEOUT_MTAG_TX_NOT_FOUND Message \(section 3.3.5.3.5.2\).](#)

3.3.1.11.3 Ended

This is the final state.

3.3.1.12 CONNTYPE_TXUSER_SETTXTIMEOUT2 Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_SETTXTIMEOUT2 \(section 2.2.8.3.4\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Set Timeout Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_SETTXTIMEOUT2 initiator states.

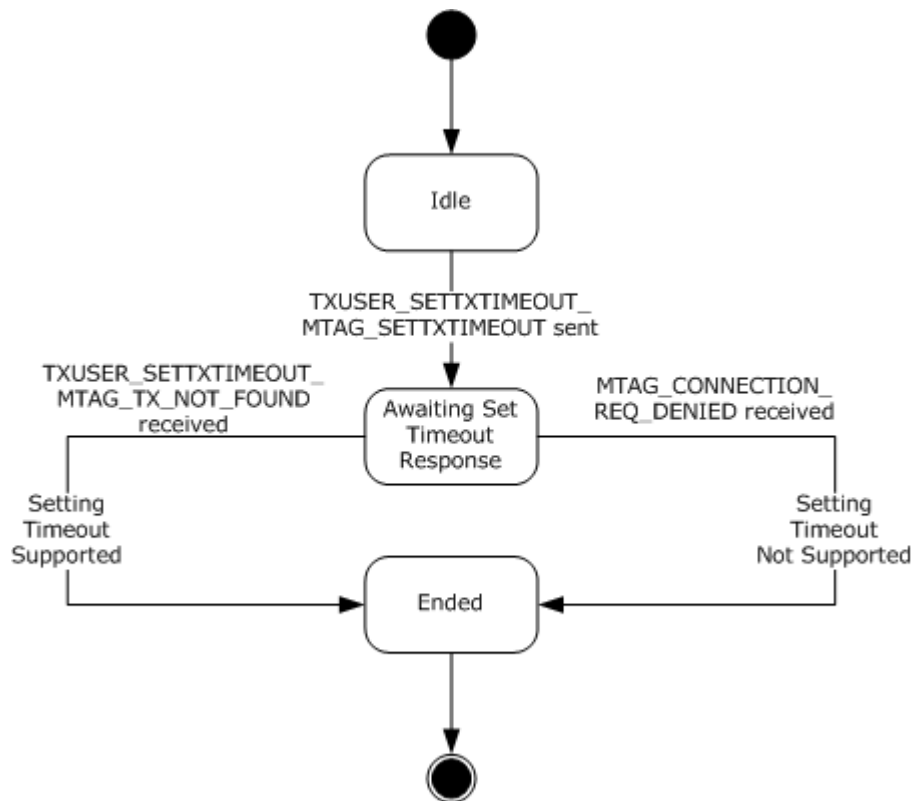


Figure 25: CONNTYPE_TXUSER_SETTXTIMEOUT2 initiator states

3.3.1.12.1 Idle

This is the initial state. The following event is processed in this state:

- [Changing a Transaction Time-out Using CONNTYPE_TXUSER_SETTXTIMEOUT2 \(section 3.3.4.2.2\)](#).

3.3.1.12.2 Awaiting Set Timeout Response

The following event is processed in this state:

- [Receiving a TXUSER SETTXTIMEOUT MTAG TX NOT FOUND Message \(section 3.3.5.3.4.1\).](#)

3.3.1.12.3 Ended

This is the final state.

3.3.1.13 CONNTYPE_TXUSER_TRACE Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_TRACE \(section 2.2.8.3.5\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Trace Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_TRACE initiator states.

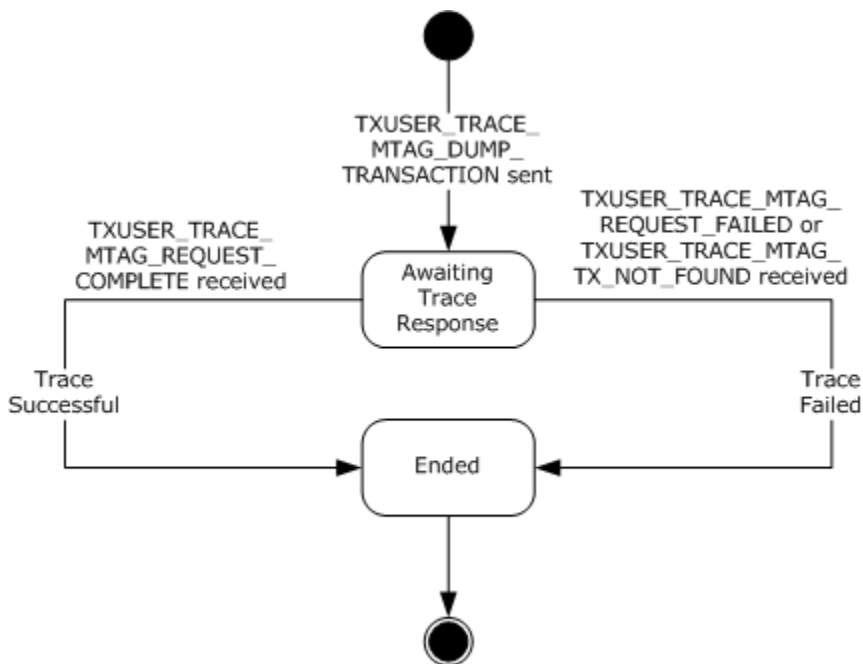


Figure 26: CONNTYPE_TXUSER_TRACE initiator states

3.3.1.13.1 Idle

This is the initial state. The following event is processed in this state:

- [Generating Trace Records for a Transaction Using CONNTYPE_TXUSER_TRACE \(section 3.3.4.4\).](#)

3.3.1.13.2 Awaiting Trace Response

The following events are processed in this state:

- [Receiving a TXUSER_TRACE_MTAG_REQUEST_COMPLETE Message \(section 3.3.5.3.5.1\).](#)

- [Receiving a TXUSER_TRACE_MTAG_REQUEST_FAILED or TXUSER_TRACE_MTAG_TX_NOT_FOUND Message \(section 3.3.5.3.5.2\)](#)

3.3.1.13.3 Ended

This is the final state.

3.3.1.14 CONNTYPE_TXUSER_GETSECURITYFLAGS Initiator States

The application MUST act as an initiator for the [CONNTYPE_TXUSER_GETSECURITYFLAGS \(section 2.2.8.4.1\)](#) connection type. In this role, the application MUST provide support for the following states:

- Idle
- Awaiting Get Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_GETSECURITYFLAGS initiator states.

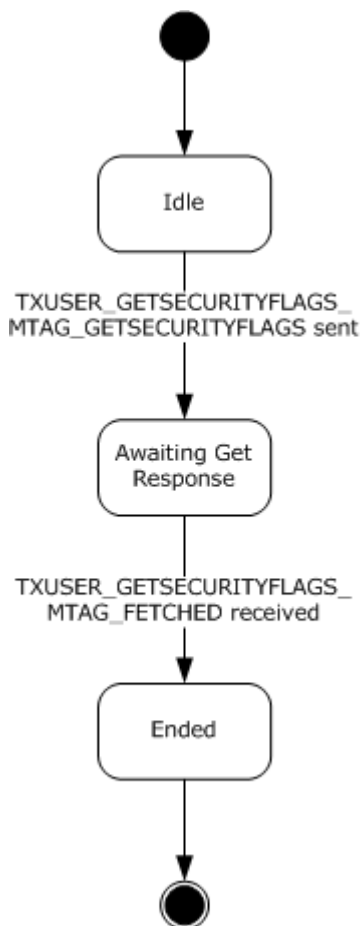


Figure 27: CONNTYPE_TXUSER_GETSECURITYFLAGS initiator states

3.3.1.14.1 Idle

This is the initial state. The following event is processed in this state:

- [Obtaining the Security Configuration of the Transaction Manager Using CONNTYPE_TXUSER_GETSECURITYFLAGS \(section 3.3.4.10\).](#)

3.3.1.14.2 Awaiting Get Response

The following event is processed in this state:

- [Receiving a TXUSER_GETSECURITYFLAGS_MTAG_FETCHED Message \(section 3.3.5.4.1.1\).](#)

3.3.1.14.3 Ended

This is the final state.

3.3.2 Timers

No timers apply here.

3.3.3 Initialization

When an application is initialized:

- The Transaction Manager **Name** field MUST be set to a value that is obtained from an implementation-specific source.

3.3.4 Higher-Layer Triggered Events

The application MUST be prepared to process a set of higher-layer events. These events are triggered by decisions that are made by the higher-layer business logic of the application. The motivations and details of the higher-layer business logic are specific to the implementation of the application and the software environment in which it executes.

When the application processes one of these events, it MUST communicate one of the following results to the higher-layer business logic:

- Success
- Failure
- Transaction Committed
- Transaction Aborted
- Transaction In Doubt

The application MUST be prepared to process the events in the following sections.

3.3.4.1 Beginning a Transaction

If the higher-layer business logic decides to begin a transaction with a predetermined transaction identifier:

- If the transaction manager of the application supports the [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#) connection type as specified in section [2.2.8](#):
 - The application MUST attempt to begin a transaction by using the CONNTYPE_TXUSER_PROMOTE (section 2.2.8.1.3) connection type.
- Otherwise:
 - The application MUST return a Failure result to the higher-layer business logic.

If the higher-layer business logic decides to begin a transaction without using a predetermined transaction identifier, the application MUST perform the following actions:

- If the transaction manager supports the [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) connection type as specified in [2.2.8](#):
 - The application MUST attempt to begin a transaction by using CONNTYPE_TXUSER_BEGIN2 (section 2.2.8.1.2).
- Otherwise:
 - The application MUST attempt to begin a transaction by using [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#)

3.3.4.1.1 Beginning a Transaction Using CONNTYPE_TXUSER_BEGIN2

The application MUST perform the following actions:

- Initiate a new [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) connection using the transaction manager **Name** field of the application.
- Add the connection to the connection list of the transaction.
- Send a [TXUSER_BEGIN2_MTAG_BEGIN \(section 2.2.8.1.2.2\)](#) message using the connection and the values that are provided by the higher-layer business logic:
 - The **isoLevel**, **dwTimeout**, **szDesc**, and **isoFlags** fields MUST be set as specified in section [2.2.8](#).
- Set the connection state to Awaiting Begin Response.

3.3.4.1.2 Beginning a Transaction Using CONNTYPE_TXUSER_BEGINNER

The application MUST perform the following actions:

- Initiate a new [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#) connection using the transaction manager Name field of the application.
- Add the connection to the transaction connection list.
- Send a [TXUSER_BEGINNER_MTAG_BEGIN \(section 2.2.8.1.1.2\)](#) message using the connection. The following message fields MUST be set to values that are provided by the higher-layer business logic:
 - The **isoLevel** field set to the required isolation-level value.
 - The **dwTimeout** field MUST be set to the required time-out value.

- The **szDesc** field MUST be set to the required **transaction description** string.
- The **isoFlags** field MUST be set to the required isolation flags value.
- Set the connection state to Awaiting Begin Response.

3.3.4.1.3 Beginning a Transaction Using CONNTYPE_TXUSER_PROMOTE

The application MUST perform the following actions:

- Initiate a new [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#) connection using the transaction manager Name field of the application.
- Add the connection to the transaction connection list, as specified in section [2.2.8](#).
- Send a [TXUSER_BEGINNER_MTAG_PROMOTE \(section 2.2.8.1.3.1\)](#) message using the connection. The following message fields MUST be set to values that are provided by the higher-layer business logic:
 - The **isoLevel** field to the wanted isolation-level value.
 - The **dwTimeout** field to the wanted time-out value.
 - The **szDesc** field to the wanted transaction description string.
 - The **isoFlags** field to the wanted isolation flags value.
 - The **guidTx** field to the wanted predetermined transaction identifier.
- Set the connection state to Awaiting Promote Response.

3.3.4.2 Changing a Transaction Timeout

If the higher-layer business logic changes the time-out of an existing transaction, the application MUST perform the following steps:

- If the application is not a root application for this transaction:
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - If the root transaction manager supports the [CONNTYPE_TXUSER_SETTXTIMEOUT2 \(section 3.3.1.12\)](#) connection type, as specified in section [2.2.8](#):
 - The application MUST attempt to change the transaction time-out by using CONNTYPE_TXUSER_SETTXTIMEOUT2 (section 3.3.1.12).
 - Otherwise:
 - The application MUST attempt to change the transaction time-out by using [CONNTYPE_TXUSER_SETTXTIMEOUT \(section 3.3.1.11\)](#).

3.3.4.2.1 Changing a Transaction Timeout Using CONNTYPE_TXUSER_SETTXTIMEOUT

The application MUST perform the following actions:

- Find an instance of a [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#) connection in the connection list of the transaction.
- If a connection is not found, the application MUST return a failure result to the higher-layer business logic.
- Otherwise, if the connection state is not [Processing Transaction \(section 3.3.1.1.3\)](#):
 - The application MUST return a failure result to the higher-layer business logic.
- Otherwise:
 - Initiate a new [CONNTYPE_TXUSER_SETTXTIMEOUT \(section 2.2.8.3.3\)](#) connection using the transaction manager **Name** field of the application.
 - Add the connection to the connection list of the transaction.
 - Send a [TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT \(section 2.2.8.1.2.7\)](#) message using the CONNTYPE_TXUSER_SETTXTIMEOUT (section 2.2.8.3.3) connection:
 - Set the **guidTx** field to the identifier for the transaction.
 - Set the dwTxTimeout value to the time-out value that is provided by the higher-layer business logic, expressed as a total number of milliseconds.
 - Set the connection state to [Awaiting Set Timeout Response \(section 3.3.1.11.2\)](#).

3.3.4.2.2 Changing a Transaction Timeout Using CONNTYPE_TXUSER_SETTXTIMEOUT2

The application MUST perform the following steps:

- Find an instance of a [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) or [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#) connection in the connection list of the transaction.
- If a connection is not found:
 - The application MUST return a failure result to the higher-layer business logic.
- Otherwise, if the connection state is not [Processing Transaction \(section 3.3.1.2.3\)](#) or [Processing Transaction \(section 3.3.1.3.3\)](#):
 - The application MUST return a failure result to the higher-layer business logic.
- Otherwise:
 - Initiate a new [CONNTYPE_TXUSER_SETTXTIMEOUT2 \(section 2.2.8.3.4\)](#) connection using the **Transaction Manager Name** field of the application.
 - Add the connection to the transaction connection list.
 - Add the following two elements to the **Connection-Specific Data** field of the new connection:
 - The transaction object.
 - The time-out value that is provided by the higher-layer business logic.

- Send a [TXUSER_SETTXTIMEOUT MTAG_SETTXTIMEOUT \(section 2.2.8.1.2.7\)](#) using the connection:
 - The **guidTx** field MUST be set to a NULL GUID.
 - The **dwTxTimeout** value MUST be set to zero.
- Set the connection state to [Awaiting Set Timeout Response \(section 3.3.1.2.4\)](#).

3.3.4.3 Creating an Export Connection

If the higher-layer business logic initiates a push propagation by using a specified [SWhereabouts](#) structure, the application MUST perform the following actions:

- Initiate a new [TXUSER_EXPORT MTAG_EXPORT \(section 2.2.8.2.2.2.6\)](#) connection by using the **Transaction Manager Name** field of the application.
- Add the connection to the **Transaction Manager Name** field.
- If the transaction manager supports the [TXUSER_EXPORT MTAG_CREATE2 \(section 2.2.8.2.2.2.2\)](#) MTAG, as specified in [OLETX_TM_ADDR \(section 2.2.4.2\)](#):
 - Send a TXUSER_EXPORT_MTAG_CREATE2 message by using the connection.
- Otherwise:
 - Send a [TXUSER_EXPORT_MTAG_CREATE \(section 2.2.8.2.2.2.1\)](#) message using the connection:
- The **SourceTmAddr** field of the message MUST be set either to an OLETX_TM_ADDR (section 2.2.4.2) structure or a [NAMEOBJECTBLOB \(section 2.2.5.3\)](#) structure, as specified in section [2.2.4.2](#)
 - Find the [STmToTmProtocol](#) entries in the SWhereabouts structure corresponding to TmProtocolMsdtcV1 and TmProtocolMsdtcV2. See section [2.2.5.11](#) for more information.
 - The **guidSignature** field MUST be set as specified in section [2.2.4.2](#).
 - If the **SourceTmAddr** field is an OLETX_TM_ADDR (section 2.2.4.2) structure, the fields of the OLETX_TM_ADDR structure MUST be set as follows:
 - The **guidSignature** field MUST be set as specified in section [2.2.4.2](#).
 - The **guidEndpoint** field MUST be set to the **guidEndpointID** field of the [SDtcCmEndpointInfoV1](#) structure.
 - The **grbComProtsSupported** field MUST be set to the **comprotSupported** field of the [SDtcCmEndpointInfoV1](#) structure.
 - If a TmProtocolMsdtcV2 entry was found:
 - The **wszHostName** field MUST be set to the **wszHostName** field of the [SDtcCmEndpointInfoV2](#) structure.
 - Otherwise:

- The **wszHostName** field MUST be set to the **szHostName** field of the SDtcCmEndpointInfoV1 structure and converted to Unicode little-endian UTF-16 encoding. This field MUST NOT contain a Unicode byte-order-mark (BOM) character.
- Otherwise, if the **SourceTmAddr** field is a NAMEOBJECTBLOB structure, the fields of the NAMEOBJECTBLOB structure MUST be set as follows:
 - The **szGuid** field MUST be set to the **guidEndpointID** field of the SDtcCmEndpointInfoV1 structure and formatted as a string as specified in [\[C706\]](#) appendix A.
 - The **grbComProtsSupported** field MUST be set to the **comprotSupported** field of the SDtcCmEndpointInfoV1 structure.
 - The **szHostName** field MUST be set to the **szHostName** field of the SDtcCmEndpointInfoV1 structure.
 - The **dwcbHostName** and **dwReserved1** fields MUST be set as specified in section [2.2.5.3](#).
- Set the connection state to Awaiting Create Response.

3.3.4.4 Generating Trace Records for a Transaction Using CONNTYPE_TXUSER_TRACE

If the higher-layer business logic specifies that transaction trace records are to be generated to the trace file of the transaction manager, the application MUST perform the following steps:

- Initiate a new [CONNTYPE_TXUSER_TRACE \(section 2.2.8.3.5\)](#) connection by using the **Transaction Manager Name** field of the application.
- Send a [TXUSER_TRACE MTAG DUMP TRANSACTION \(section 2.2.8.3.5.1\)](#) message:
 - The **guidTx** field MUST be set to the identifier of the provided transaction.
- Set the connection state to Awaiting Trace Response.

3.3.4.5 Importing a Transaction

If the higher-layer business logic specifies that a transaction be imported by using an [StxInfo \(section 2.2.5.10\)](#) structure, the application MUST perform the following steps:

- If the transaction manager of the application supports the [CONNTYPE_TXUSER_IMPORT2](#) connection type as specified in section [2.2.1.1.1](#):
 - The application MUST attempt to import the transaction time-out by using CONNTYPE_TXUSER_IMPORT2 (section 2.2.8.2.2.4).
- Otherwise:
 - The application MUST attempt to import the transaction by using [CONNTYPE_TXUSER_IMPORT \(section 2.2.8.2.2.3\)](#).

3.3.4.5.1 Importing a Transaction Using CONNTYPE_TXUSER_IMPORT

The application MUST perform the following actions:

- Initiate a new [CONNTYPE_TXUSER_IMPORT \(section 2.2.8.2.2.3\)](#) connection using the **Transaction Manager Name** field of the application.

- Add the connection to the transaction connection list.
- Send a [TXUSER_IMPORT_MTAG_IMPORT \(section 2.2.8.2.2.3.3\)](#) message using the connection:
 - The **guidTx** field MUST be set to the **uowTx** field of the provided [STxInfo \(section 2.2.5.10\)](#) structure.
- Set the connection state to Awaiting Import Response.

3.3.4.5.2 Importing a Transaction Using CONNTYPE_TXUSER_IMPORT2

The application MUST perform the following actions:

- Initiate a new [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.2.4\)](#) connection using the **Transaction Manager Name** field of the application.
- Add the connection to the transaction connection list.
- Send a [TXUSER_IMPORT2_MTAG_IMPORT \(section 2.2.8.2.2.4.2\)](#) message using the connection:
 - The **guidTx** field MUST be set to the **uowTx** field of the provided [STxInfo \(section 2.2.5.10\)](#) structure.
- Set the connection state to Awaiting Import Response.

3.3.4.6 Importing a Transaction with Additional Transaction Attributes

If the higher-layer business logic specifies that a transaction be imported by using a [STxInfo \(section 2.2.5.10\)](#) structure and that additional transaction attributes be set, the application MUST perform the following steps:

- If the transaction manager of the application does not support the [CONNTYPE_TXUSER_IMPORT2](#) connection type, as specified in section [2.2.1.1.1](#).
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - Send a [TXUSER_IMPORT2_MTAG_IMPORT_WITH_SET \(section 2.2.8.2.2.4.3\)](#) message, using the connection:
 - The **guidTx** field MUST be set to the **uowTx** field of the provided STxInfo (section 2.2.5.10) structure, as specified in [2.2.5.10](#).
 - The **isoLevel** field MUST be set to the provided isolation-level value.
 - The **isoFlags** field MUST be set to the provided isolation flags value.
 - The **szDesc** field MUST be set to the provided description string.
 - Set the connection state to [Awaiting Import Response \(section 3.3.1.6.2\)](#).

3.3.4.7 Initiating Transaction Commit

If the higher-layer business logic initiates the commit of an existing transaction, the application MUST perform the following steps:

- Find a [CONNTYPE_TXUSER_PROMOTE](#), [CONNTYPE_TXUSER_BEGINNER](#), or [CONNTYPE_TXUSER_BEGIN2](#) connection in the transaction connection list.
- If a CONNTYPE_TXUSER_PROMOTE is found:
 - The application MUST attempt to complete the transaction by using CONNTYPE_TXUSER_PROMOTE.
- Otherwise, if a CONNTYPE_TXUSER_BEGINNER is found:
 - The application MUST attempt to complete the transaction by using CONNTYPE_TXUSER_BEGINNER.
- Otherwise, if a CONNTYPE_TXUSER_BEGIN2 is found:
 - The application MUST attempt to complete the transaction by using CONNTYPE_TXUSER_BEGIN2.
- Otherwise:
 - The application MUST return a failure result to the higher-layer business logic.

3.3.4.7.1 Commit a Transaction Using CONNTYPE_TXUSER_BEGIN2

The application MUST perform the following actions:

- If the connection state is not Processing Transaction:
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - Send a [TXUSER_BEGIN2_MTAG_COMMIT \(section 2.2.8.1.2.3\)](#) message using the [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) connection:
 - Set the connection state to Awaiting Commit Response.

3.3.4.7.2 Commit a Transaction Using CONNTYPE_TXUSER_BEGINNER

The application MUST perform the following actions:

- If the state of the connection is not Processing Transaction:
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - Send a [TXUSER_BEGINNER_MTAG_COMMIT \(section 2.2.8.1.1.6\)](#) message using the [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#) connection:
 - The **grfRM** field MUST be set to 0.
 - The **fAsyncFull** field MUST be set to 0.
 - Set the connection state to Awaiting Commit Response.

3.3.4.7.3 Commit a Transaction Using CONNTYPE_TXUSER_PROMOTE

The application MUST perform the following actions:

- If the state of the connection is not Processing Transaction:
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - Send a [TXUSER_BEGIN2 MTAG COMMIT \(section 2.2.8.1.2.3\)](#) message using the [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#) connection:
 - The **grfRM** field MUST be set to 0.
 - Set the connection state to Awaiting Commit Response.

3.3.4.8 Initiating Transaction Rollback

If the higher-layer business logic initiates the rollback of an existing transaction, the application MUST perform the following steps:

- Find a [CONNTYPE_TXUSER_PROMOTE](#), [CONNTYPE_TXUSER_BEGIN2](#), [CONNTYPE_TXUSER_BEGINNER](#), [CONNTYPE_TXUSER_IMPORT2](#), or [CONNTYPE_TXUSER_IMPORT](#) connection in the transaction connection list.
- If a CONNTYPE_TXUSER_PROMOTE is found:
 - The application MUST attempt to roll back a transaction by using CONNTYPE_TXUSER_PROMOTE.
- Otherwise, if a CONNTYPE_TXUSER_BEGIN2 is found:
 - The application MUST attempt to roll back a transaction by using CONNTYPE_TXUSER_BEGIN2.
- Otherwise, if a CONNTYPE_TXUSER_BEGINNER is found:
 - The application MUST attempt to roll back a transaction by using CONNTYPE_TXUSER_BEGINNER
- Otherwise, if a CONNTYPE_TXUSER_IMPORT2 is found:
 - The application MUST attempt to roll back a transaction by using CONNTYPE_TXUSER_IMPORT2
- Otherwise, if a CONNTYPE_TXUSER_IMPORT is found:
 - The application MUST attempt to roll back a transaction by using CONNTYPE_TXUSER_IMPORT
- Otherwise, the application MUST return a failure result to the higher-layer business logic.

3.3.4.8.1 Abort a Transaction Using CONNTYPE_TXUSER_BEGIN2

The application MUST perform the following actions:

- If the connection state is not Processing Transaction:

- Return a failure result to the higher-layer business logic.
- Otherwise:
 - Send a [TXUSER_BEGIN2_MTAG_ABORT \(section 2.2.8.1.2.1\)](#) message using the [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) connection.
 - Set the connection state to Awaiting Abort Response.

3.3.4.8.2 Abort a Transaction Using CONNTYPE_TXUSER_BEGINNER

The application MUST perform the following actions:

- If the connection state is not Processing Transaction:
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - Send a [TXUSER_BEGINNER_MTAG_ABORT \(section 2.2.8.1.1.1\)](#) message using the [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#) connection:
 - The **guidReason** field MUST be set to the value that is provided by the higher-layer business logic, as specified in section [2.2.8.1.1.1](#).
 - Set the connection state to Awaiting Abort Response.

3.3.4.8.3 Abort a Transaction Using CONNTYPE_TXUSER_IMPORT

The application MUST perform the following actions:

- If the connection state is not Transaction Import Successful:
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - Send a [TXUSER_IMPORT_MTAG_ABORT \(section 2.2.8.2.3.1\)](#) message using the [CONNTYPE_TXUSER_IMPORT \(section 2.2.8.2.3\)](#) connection:
 - The **guidReason** field MUST be set to the value that is provided by the higher-layer business logic, as specified in section [2.2.8.2.3.1](#).
 - Set the connection state to Awaiting Abort Response.

3.3.4.8.4 Abort a Transaction Using CONNTYPE_TXUSER_IMPORT2

The application MUST perform the following actions:

- If the connection state is not Transaction Import Successful:
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - Send a [TXUSER_IMPORT2_MTAG_ABORT \(section 2.2.8.2.4.1\)](#) message using the [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.4\)](#) connection:

- Set the connection state to Awaiting Abort Response.

3.3.4.8.5 Roll Back a Transaction Using CONNTYPE_TXUSER_PROMOTE

The application MUST perform the following actions:

- If the connection state is not Processing Transaction:
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - Send a [TXUSER_BEGIN2_MTAG_ABORT \(section 2.2.8.1.2.1\)](#) message using the [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#) connection.
 - Set the connection state to Awaiting Abort Response.

3.3.4.9 Obtaining Extended Whereabouts Using CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS

If the higher-layer business logic wants to obtain extended whereabouts for a transaction manager, the application MUST perform the following actions:

- If the transaction manager supports the connection type [CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS](#) as specified in section [2.2.8.2.2.1](#):
 - Initiate a new [CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS \(section 2.2.8.2.2.1\)](#) connection using the **Transaction Manager Name** field of the application.
 - Add the connection to the connection list of the transaction.
 - Send a [TXUSER_EXTENDEDWHEREABOUTS_MTAG_GET \(section 2.2.8.2.2.1.1\)](#) message using the connection.
 - Set the connection state to Awaiting Get Response.
- Otherwise:
 - The application MUST return a failure result to the higher-layer business logic.

3.3.4.10 Obtaining the Security Configuration of the Transaction Manager Using CONNTYPE_TXUSER_GETSECURITYFLAGS

If the higher-layer business logic wants to obtain the security configuration of the transaction manager, the application MUST perform the following steps:

- Initiate a new [CONNTYPE_TXUSER_GETSECURITYFLAGS \(section 2.2.8.4.1\)](#) connection by using the **Transaction Manager Name** field of the application.
- Send a [TXUSER_GETSECURITYFLAGS_MTAG_GETSECURITYFLAGS \(section 2.2.8.4.1.2\)](#) message.
- Set the connection state to Awaiting Get Response.

3.3.4.10.1 Obtaining the Details for a Transaction

If the higher-layer business logic wants to obtain the details for a transaction, the application **MUST** perform the following steps:

- Initiate a new [CONNTYPE_TXUSER_GETTXDETAILS \(section 2.2.8.3.1\)](#) connection by using the **Transaction Manager Name** field of the application.
- Add the connection to the connection list of the transaction.
- Send a [TXUSER_GETTXDETAILS_MTAG_GET \(section 2.2.8.3.1.1\)](#) message using the connection:
 - The **guidTx** field **MUST** be set to the provided transaction identifier.
- Set the connection state to Awaiting Response.

3.3.4.11 Pulling a Transaction

If the higher-layer business logic wants to perform pull-based propagation of a transaction by using a [Propagation Token](#) structure, the application **MUST** perform the following actions:

- Initiate a new [CONNTYPE_TXUSER_ASSOCIATE](#) connection using the **Transaction Manager Name** field of the application.
- Create a new transaction object that uses the **guidTx** field of the [Propagation Token](#) as the transaction identifier.
- Add the connection to the connection list of the transaction object.
- Set the **Connection-Specific Data** field of the connection to reference the transaction object.
- Send a [TXUSER_ASSOCIATE_MTAG_ASSOCIATE](#) message using the connection:
 - The **guidTx** field **MUST** be set to the **guidTx** field of the [Propagation Token](#).
 - The **isoLevel** field **MUST** be set to the **isoLevel** field of the [Propagation Token](#).
 - The **isoFlags** field **MUST** be set to the **isoFlags** field of the [Propagation Token](#).
 - The **szDesc** field **MUST** be set to the **szDesc** field of the [Propagation Token](#).
 - The **SourceTmAddr** field in the message **MUST** be set from either an [OLETX_TM_ADDR](#) structure or a [NAMEOBJECTBLOB](#) structure, as specified in section [2.2.1.1.1](#):
 - If the **SourceTmAddr** field is an [OLETX_TM_ADDR](#) structure, the [OLETX_TM_ADDR](#) structure fields **MUST** be set as follows:
 - The **guidSignature** field **MUST** be set as specified in section [2.2.4.2](#).
 - The **guidEndpoint** field **MUST** be set to the [Propagation Token](#)'s [NameObject](#) field's **szGuid** field, converted from a string to a GUID as specified in [\[C706\]](#) appendix A.
 - The **grbComProtsSupported** field **MUST** be set to the [Propagation Token](#)'s [NameObject](#) field's **grbComProtsSupported** field.
 - If the **dwVersionMax** field of the [Propagation Token](#) is at least 2:

- The **wszHostName** field MUST be set to the Propagation_Token's NameObject field's **wszHostName** field.
- Otherwise:
 - The **wszHostName** field MUST be set to the Propagation_Token's **NameObject** field's **szHostName** field, converted to little-endian UTF-16 encoding. This field MUST NOT contain a Unicode byte-order-mark (BOM) character.
 - Otherwise, if the **SourceTmAddr** field is a NAMEOBJECTBLOB structure, the NAMEOBJECTBLOB structure fields MUST be set to the same values as the NameObject structure of the Propagation_Token.
- The **cbSourceTmAddr** field MUST be set as specified in section [2.2.8.2.1.1.1](#).
- Set the connection state to Awaiting Associate Response.

3.3.4.12 Push a Transaction Using an Existing Export Connection

If the higher-layer business logic decides to export a transaction by using an existing export connection, the application MUST perform the following actions:

- If the provided connection state is not Connection Active:
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - Send a [TXUSER_EXPORT_MTAG_EXPORT \(section 2.2.8.2.2.6\)](#) message using the provided connection:
 - The **guidTX** field MUST be set to the provided **Identifier** field of the transaction object.
 - Set the connection state to Awaiting Export Response.

3.3.4.13 Resolving a Transaction

If the higher-layer business logic determines that it needs to manually resolve the outcome of a transaction, the application MUST perform the following steps:

- If the transaction is not in either the [Failed to Notify \(section 3.2.1.4.13\)](#) or the [In Doubt \(section 3.2.1.4.12\)](#) state:
 - Return a failure result to the higher-layer business logic.
- Otherwise:
 - Initiate a new [CONNTYPE_TXUSER_RESOLVE \(section 2.2.8.3.2\)](#) connection using the **Transaction Manager Name** field of the application.
 - If the transaction is in the Failed to Notify (section 3.2.1.4.13) state:
 - Send a [TXUSER_RESOLVE_MTAG_FORGET_COMMITTED \(section 2.2.8.3.2.5\)](#) message using the connection:
 - The **guidTx** field MUST be set to the identifier of the provided transaction.
 - Set the connection state to [Awaiting Forget Response \(section 3.3.1.10.3\)](#).

- Otherwise, if the transaction is in the In Doubt (section 3.2.1.4.12) state:
 - If the higher-layer business logic wants to manually resolve the transaction outcome as Commit:
 - Send a [TXUSER_RESOLVE_MTAG_CHILD_COMMIT \(section 2.2.8.3.2.3\)](#) message using the connection:
 - The **guidTx** field MUST be set to the identifier of the provided transaction.
 - Set the connection state to [Awaiting Commit Response \(section 3.3.1.1.4\)](#).
 - Otherwise, if the higher-layer business logic wants to manually resolve the transaction outcome as Abort:
 - Send a [TXUSER_RESOLVE_MTAG_CHILD_ABORT \(section 2.2.8.3.2.2\)](#) message using the connection:
 - The **guidTx** field MUST be set to the identifier of the provided transaction.
 - Set the connection state to [Awaiting Abort Response \(section 3.3.1.1.5\)](#).

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Transaction Initiation and Completion

3.3.5.1.1 CONNTYPE_TXUSER_BEGINNER as Initiator

For all messages that are received in this connection type, the application MUST process the message as specified in section [3.1](#). The application MUST also follow the processing rules that are specified in the following sections.

3.3.5.1.1.1 Receiving a TXUSER_BEGINNER_MTAG_BEGUN Message

When the application receives a [TXUSER_BEGINNER_MTAG_BEGUN](#) message, the application MUST perform the following actions:

- If the connection state is [Awaiting Begin Response](#):
 - Set the connection state to Processing Transaction.
 - Create a transaction object that is initialized as follows:
 - Set the transaction **Identifier** field to the **guidTx** field from the message.
 - Add the connection to the connection list of the transaction.
 - Set the **Connection-Specific Data** field of the connection to the transaction object.
 - Return a success result and a reference to the transaction object to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.1.1.2 Receiving a TXUSER_BEGINNER_MTAG_BEGIN_NO_MEM or TXUSER_BEGINNER_MTAG_BEGIN_LOG_FULL Message

When the application receives either of these messages, the application MUST perform the following actions:

- If the connection state is Awaiting Timeout Set Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.1.1.3 Receiving a TXUSER_BEGINNER_MTAG_REQUEST_COMPLETED Message

When the application receives a [TXUSER_BEGINNER_MTAG_REQUEST_COMPLETED](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Commit Response:
 - Return a Transaction Committed result to the higher-layer business logic.
 - Set the connection state to Ended.
- If the connection state is Awaiting Abort Response:
 - Return a Transaction Aborted result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.1.1.4 Receiving a TXUSER_BEGINNER_MTAG_COMMIT_TOO_LATE Message

When the application receives a [TXUSER_BEGINNER_MTAG_COMMIT_TOO_LATE](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Commit Response:
 - Return a Transaction Aborted result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.1.1.5 Receiving a TXUSER_BEGINNER_MTAG_COMMIT_INDOUBT Message

When the application receives a [TXUSER_BEGINNER_MTAG_COMMIT_INDOUBT](#) (section 2.2.8.1.1.7) message, the application MUST perform the following actions:

- If the connection state is Awaiting Commit Response:
 - Return a transaction [In Doubt](#) (section 3.2.1.4.12) result to the higher-layer business logic.
 - Set the connection state to [Ended](#) (section 3.2.1.4.14).

- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.1.1.6 Connection Disconnected

When a [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Begin Response:
 - Return a failure result to the higher-layer business logic.
- If the connection state is in the Awaiting Abort Response state:
 - Return a Transaction Aborted result to the higher-layer business logic.
- If the connection state is Awaiting Commit Response:
 - Return a transaction [In Doubt \(section 3.2.1.4.12\)](#) result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.5.1.2 CONNTYPE_TXUSER_BEGIN2 as Initiator

For all messages that are received in this connection type, the application MUST process the message as specified in section [3.1](#). The application MUST also follow the processing rules that are specified in the following sections.

3.3.5.1.2.1 Receiving a TXUSER_BEGIN2_MTAG_SINK_BEGUN Message

When the application receives a [TXUSER_BEGIN2_MTAG_SINK_BEGUN](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Begin Response, the application MUST:
 - Set the connection state to Processing Transaction.
 - Create a transaction object that is initialized as follows:
 - Set the transaction **Identifier** field to the **guidTx** field from the message.
 - Add the connection to the transaction connection list.
 - Set the transaction field of the connection to the transaction.
 - Return a success result and a reference to the transaction object to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.1.2.2 Receiving a TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE Message

When the application receives a [TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Timeout Set Response, the application MUST:

- Set the connection state to Processing Transaction.
- Return a success result to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.1.2.3 Receiving a TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE Message

When the application receives a [TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Timeout Set Response, the application MUST:
 - Set the connection state to Processing Transaction.
 - Return a failure result to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.1.2.4 Receiving a TXUSER_BEGIN2_MTAG_SINK_ERROR Message

When the application receives a [TXUSER_BEGIN2_MTAG_SINK_ERROR](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Begin Response:
 - If the **Error** field in the message is set to TRUN_TXBEGIN_ERROR_NO_MEM or TRUN_TXBEGIN_ERROR_BEGIN_LOG_FULL:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
 - Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).
- If the connection state is Awaiting Commit Response:
 - If the **Error** field in the message is set to TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED or TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED or TRUN_TXBEGIN_ERROR_NOTIFY_INDOUBT:
 - Return the corresponding transaction outcome as a result to the higher-layer business logic.
 - Set the connection state to Ended.
 - Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).
- If the connection state is Awaiting Abort Response:
 - If the **Error** field of the message is set to TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED:
 - Return a transaction aborted result to the higher-layer business logic.
 - Set the connection state to Ended.

- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).
- If the Connection state is Awaiting Set Timeout Response:
 - If the **Error** field of the message is set to TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED:
 - Return a Transaction Aborted result to the higher-layer business logic.
 - Set the connection state to Ended.
 - Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.1.2.5 Connection Disconnected

When a [CONNTYPE_TXUSER_BEGIN2](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Begin Response:
 - Return a failure result to the higher-layer business logic.
- If the connection state is in the Awaiting Timeout Set Response state, the Processing Transaction state or the Awaiting Abort Response state:
 - Return a transaction aborted result to the higher-layer business logic.
- If the connection state is Awaiting Commit Response:
 - Return a transaction [In Doubt \(section 3.2.1.4.12\)](#) result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.5.1.3 CONNTYPE_TXUSER_PROMOTE as Initiator

The [CONNTYPE_TXUSER_PROMOTE](#) connection type that is acting as an initiator MUST follow the same message processing rules as the [CONNTYPE_TXUSER_BEGIN2](#) connection type that is acting as an initiator.

3.3.5.2 Transaction Propagation

3.3.5.2.1 Pull Propagation

3.3.5.2.1.1 CONNTYPE_TXUSER_ASSOCIATE as Initiator

For all messages that are received in this connection type, the application MUST process the messages as specified in section [3.1](#).

The application MUST also follow the processing rules that are specified in the following sections.

3.3.5.2.1.1.1 Receiving a TXUSER_ASSOCIATE_MTAG_ASSOCIATED Message

When the application receives a [TXUSER_ASSOCIATE_MTAG_ASSOCIATED](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Associate Response, the application MUST:
 - Set the connection state to [Active \(section 3.4.1.4.3\)](#).
 - Return a success result and a reference to the transaction object that is referenced by this connection's Connection-Specific Data field to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.1.1.2 Receiving Other TXUSER_ASSOCIATE_MTAG Messages

When the application receives one of these messages:

- [TXUSER_ASSOCIATE_MTAG_TX_NOT_FOUND](#)
- [TXUSER_ASSOCIATE_MTAG_TOO_LATE](#)
- [TXUSER_ASSOCIATE_MTAG_CREATE_BAD_TMADDR](#)
- [TXUSER_ASSOCIATE_MTAG_LOG_FULL_LOCAL](#)
- [TXUSER_ASSOCIATE_MTAG_NO_MEM_LOCAL](#)
- [TXUSER_ASSOCIATE_MTAG_LOG_FULL_REMOTE](#)
- [TXUSER_ASSOCIATE_MTAG_NO_MEM_REMOTE](#)
- [TXUSER_ASSOCIATE_MTAG_TOO_MANY_REMOTE](#)

the application MUST perform the following actions:

- If the connection state is Awaiting Associate Response:
 - Set the connection state to Ended.
 - Return a failure result to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.1.1.3 Receiving a TXUSER_IMPORT2_MTAG_SINK_ERROR Message

When the application receives a TXUSER_IMPORT2_MTAG_SINK_ERROR message, the application MUST perform the following actions:

- If the connection state is Active:
 - If the **Error** field in the message is set to TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED or TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED or TRUN_TXBEGIN_ERROR_NOTIFY_INDOUBT:
 - Notify the higher-layer business logic of the outcome of the transaction.
 - Set the connection state to Ended.
 - Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.1.1.4 Connection Disconnected

When a [CONNTYPE_TXUSER_ASSOCIATE \(section 2.2.8.2.1.1\)](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Associate Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.5.2.2 Push Propagation

3.3.5.2.2.1 CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS as Initiator

For all messages that are received in this connection type, the application MUST process the message as specified in section [3.1](#). The application MUST also follow the processing rules as specified in the following sections.

3.3.5.2.2.1.1 Receiving a TXUSER_EXTENDEDWHEREABOUTS_MTAG_GOT Message

When the application receives a [TXUSER_EXTENDEDWHEREABOUTS_MTAG_GOT](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Get Response:
 - Return a success result and the extended whereabouts data to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.1.2 Receiving a TXUSER_EXTENDEDWHEREABOUTS_MTAG_NOMEM Message

When the application receives a [TXUSER_EXTENDEDWHEREABOUTS_MTAG_NOMEM](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Get Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.1.3 CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS Connection Disconnected

When a [CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS \(section 2.2.8.2.2.1\)](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Get Response:
 - Return a failure result to the higher-layer business logic.

- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.5.2.2.2 CONNTYPE_TXUSER_EXPORT as Initiator

For all messages that are received in this connection type, the application MUST process the message as specified in section [3.1](#). The application MUST also follow the processing rules as specified in the following sections.

3.3.5.2.2.2.1 Receiving a TXUSER_EXPORT_MTAG_CREATED Message

When the application receives a [TXUSER_EXPORT_MTAG_CREATED](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Create Response:
 - Set the connection state to Connection Active.
 - Return a success result to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.2.2 Receiving a TXUSER_EXPORT_MTAG_CREATE_BAD_TMADDR or TXUSER_EXPORT_MTAG_CREATE_NET_TX_DISABLED Message

When the application receives one of these messages, the application MUST perform the following actions:

- If the connection state is Awaiting Create Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.2.3 Receiving a TXUSER_EXPORT_MTAG_EXPORTED Message

When the application receives one of these messages, the application MUST perform the following actions:

- If the connection state is Awaiting Export Response:
 - Set the connection state to Connection Active.
 - Return a success result to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.2.4 Receiving a TXUSER_EXPORT_MTAG_EXPORT_LOG_FULL, TXUSER_EXPORT_MTAG_EXPORT_NO_MEM, TXUSER_EXPORT_MTAG_EXPORT_TOO_LATE, TXUSER_EXPORT_MTAG_EXPORT_TOO_MANY, or TXUSER_EXPORT_MTAG_EXPORT_NOT_FOUND Message

When the application receives one of these messages, the application MUST perform the following actions:

- If the connection state is Awaiting Export Response, the application MUST:
 - Set the connection state to Connection Active.
 - Return a failure result to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.2.5 CONNTYPE_TXUSER_EXPORT Connection Disconnected

When a [CONNTYPE_TXUSER_EXPORT \(section 2.2.8.2.2.2\)](#) connection is disconnected, the application MUST perform the following additional actions:

- If the connection state is Awaiting Create Response or Awaiting Export Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.5.2.2.3 CONNTYPE_TXUSER_IMPORT as Initiator

For all messages that are received in this connection type, the application MUST process the messages as specified in section [3.1](#). The application MUST also follow the processing rules as specified in the following sections.

3.3.5.2.2.3.1 Receiving a TXUSER_IMPORT_MTAG_IMPORTED Message

When the application receives a [TXUSER_IMPORT_MTAG_IMPORTED](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Import Response:
 - Set the connection state to Transaction Import Successful.
 - Return a success result and a reference to the transaction object to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.3.2 Receiving a TXUSER_IMPORT_MTAG_IMPORT_TX_NOT_FOUND OR TXUSER_IMPORT_MTAG_ABORT_TOO_LATE Message

When the application receives either a [TXUSER_IMPORT_MTAG_IMPORT_TX_NOT_FOUND](#) or [TXUSER_IMPORT_MTAG_ABORT_TOO_LATE](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Import Response:
 - Set the connection state to Ended.
 - Return a failure result to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.3.3 Receiving a TXUSER_IMPORT_MTAG_REQUEST_COMPLETED Message

When the application receives a [TXUSER_IMPORT_MTAG_REQUEST_COMPLETED](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Abort Response:
 - Set the connection state to Ended.
 - Return a success result to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.3.4 Connection Disconnected

When a [CONNTYPE_TXUSER_IMPORT \(section 2.2.8.2.2.3\)](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Import Response or Awaiting Abort Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.5.2.2.4 CONNTYPE_TXUSER_IMPORT2 as Initiator

For all messages that are received in this connection type, the application MUST process the messages as specified in section [3.1](#). The application MUST also follow the processing rules as specified in the following sections.

3.3.5.2.2.4.1 Receiving a TXUSER_IMPORT2_MTAG_IMPORTED Message

When the application receives a [TXUSER_IMPORT_MTAG_IMPORTED](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Import Response:
 - Set the connection state to Transaction Import Successful.
 - Return a success result and a reference to the transaction object to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.4.2 Receiving a TXUSER_IMPORT2_MTAG_SINK_ERROR Message

When the application receives a [TXUSER_IMPORT2_MTAG_SINK_ERROR](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Import Response:
 - If the **Error** field in the message is set to TRUN_TXIMPORT_ERROR_IMPORT_TX_NOT_FOUND:
 - Return a failure result to the higher-layer business logic.

- Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).
- If the connection state is Awaiting Abort Response:
 - If the **Error** field in the message is set to TRUN_TXIMPORT_ERROR_IMPORT_TX_NOT_FOUND:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
 - Otherwise, if the **Error** field in the message is set to TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED, TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED, or TRUN_TXBEGIN_ERROR_NOTIFY_INDOUBT:
 - Return the respective transaction outcome as a result to the higher-layer business logic.
 - Set the connection state to Ended.
 - Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).
- If the connection state is Transaction Import Successful:
 - If the **Error** field in the message is set to TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED or TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED or TRUN_TXBEGIN_ERROR_NOTIFY_INDOUBT:
 - Notify the higher-layer business logic of the outcome of the transaction.
 - Set the connection state to Ended.
 - Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.2.2.4.3 CONNTYPE_TXUSER_IMPORT2 Connection Disconnected

When a [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.2.4\)](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Import Response or Awaiting Abort Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.5.3 Transaction Administration

3.3.5.3.1 CONNTYPE_TXUSER_GETTXDETAILS as Initiator

For all messages that are received in this connection type, the application MUST process the message as specified in section [3.1](#). The application MUST also follow the processing rules as specified in the following sections.

3.3.5.3.1.1 Receiving a TXUSER_GETTXDETAILS_MTAG_GOTIT Message

When the application receives a [TXUSER_GETTXDETAILS_MTAG_GOTIT](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Response:
 - Return a success result to the higher-layer business logic, including the details that are provided in the following message fields:
 - The **vszSuperiorName** field.
 - The **vszSuperiorID** field.
 - The **ISubordinateCount** field.
 - The **rgSubordinates** field.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.3.1.2 Receiving a TXUSER_GETTXDETAILS_MTAG_TX_NOT_FOUND Message

When the application receives a [TXUSER_GETTXDETAILS_MTAG_TX_NOT_FOUND](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.3.1.3 CONNTYPE_TXUSER_GETTXDETAILS Connection Disconnected

When a [CONNTYPE_TXUSER_GETTXDETAILS \(section 2.2.8.3.1\)](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.5.3.2 CONNTYPE_TXUSER_RESOLVE as Initiator

For all messages that are received in this connection type, the application MUST process the message as specified in section [3.1.1](#). The application MUST also follow the processing rules as specified in the following sections.

3.3.5.3.2.1 Receiving a TXUSER_RESOLVE_MTAG_REQUEST_COMPLETE Message

When the application receives a [TXUSER_RESOLVE_MTAG_REQUEST_COMPLETE](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Abort Response, Awaiting Commit Response, or Awaiting Forget Response:
 - Return a success result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#)

3.3.5.3.2.2 Receiving a TXUSER_RESOLVE_MTAG_ACCESSDENIED or TXUSER_RESOLVE_MTAG_TX_NOT_FOUND Message

When the application receives one of these messages, the application MUST perform the following actions:

- If the connection state is Awaiting Abort Response, Awaiting Commit Response, or Awaiting Forget Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#)

3.3.5.3.2.3 Receiving a TXUSER_RESOLVE_MTAG_CHILD_NOT_PREPARED Message

When the application receives a [TXUSER_RESOLVE_MTAG_CHILD_NOT_PREPARED](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Abort Response or Awaiting Commit Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#)

3.3.5.3.2.4 Receiving a TXUSER_RESOLVE_MTAG_FORGET_TX_NOT_COMMITTED Message

When the application receives a [TXUSER_RESOLVE_MTAG_FORGET_TX_NOT_COMMITTED](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Forget Response:
 - Return a success result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#)

3.3.5.3.2.5 Connection Disconnected

When a [CONNTYPE_TXUSER_RESOLVE](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Abort Response, Awaiting Commit Response, or Awaiting Forget Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, the event MUST be processed as an invalid message as specified in section [3.1.8.1](#).

3.3.5.3.3 CONNTYPE_TXUSER_SETTXTIMEOUT as Initiator

For all messages that are received in this connection type, the application MUST process the message as specified in section [3.1.1](#). The application MUST also follow the processing rules as specified in the following sections.

3.3.5.3.3.1 Receiving a TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE Message

When the application receives a [TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Set Timeout Response:
 - Return a success result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.3.3.2 Receiving a TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE or TXUSER_SETTXTIMEOUT_MTAG_TX_NOT_FOUND Message

When the application receives a [TXUSER_GETTXDETAILS_MTAG_TOO_LATE](#) or [TXUSER_GETTXDETAILS_MTAG_TX_NOT_FOUND](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Set Timeout Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.3.3.3 Connection Disconnected

When a [CONNTYPE_TXUSER_SETTXTIMEOUT](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Set Timeout Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.5.3.4 CONNTYPE_TXUSER_SETTXTIMEOUT2 as Initiator

For all messages that are received in this connection type, the application MUST process the message as specified in section [3.1.1](#). The application MUST also follow the processing rules as specified in the following sections.

3.3.5.3.4.1 Receiving a TXUSER_SETTXTIMEOUT_MTAG_TX_NOT_FOUND Message

When the application receives a [TXUSER_SETTXTIMEOUT_MTAG_TX_NOT_FOUND](#) (section [2.2.8.3.3.1](#)) message, the application MUST perform the following actions:

- If the connection state is Awaiting Set Timeout Response:
 - Find an instance of a [CONNTYPE_TXUSER_BEGIN2](#) (section [2.2.8.1.2](#)) or [CONNTYPE_TXUSER_PROMOTE](#) (section [2.2.8.1.3](#)) connection in the transaction connection list. This connection is referred to as the beginner connection.
 - If a beginner connection is not found:
 - The application MUST return a failure result to the higher-layer business logic.
 - Otherwise, if the beginner connection state is not Awaiting Set Timeout Response:
 - The application MUST return a failure result to the higher-layer business logic.
 - Otherwise:
 - Send a [TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT](#) (section [2.2.8.1.2.7](#)) using the connection:
 - Set the **guidTx** field to the **Identifier** field on the transaction object that was stored in the [CONNTYPE_TXUSER_SETTXTIMEOUT2](#) (section [2.2.8.3.4](#)) **Connection-Specific Data** field of the connection. See section [3.3.4.2.2](#) for more information.
 - Set the **dwTxTimeout** field to the timeout value that was stored in the [CONNTYPE_TXUSER_SETTXTIMEOUT2](#) (section [2.2.8.3.4](#)) **Connection-Specific Data** field of the connection. See section [3.3.4.2.2](#) for more information.
 - Set the beginner connection state to Awaiting Set Timeout Response.
 - Set the [CONNTYPE_TXUSER_SETTXTIMEOUT2](#) (section [2.2.8.3.4](#)) connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in [Connection Disconnected](#) (section [3.1.8.1](#)).

3.3.5.3.4.2 Connection Disconnected

When a [CONNTYPE_TXUSER_SETTXTIMEOUT2](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Set Timeout Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#)

3.3.5.3.5 CONNTYPE_TXUSER_TRACE as Initiator

For all messages that are received in this connection type, the application MUST process the message as specified in section [3.1](#). The application MUST also follow the processing rules as specified in the following sections.

3.3.5.3.5.1 Receiving a TXUSER_TRACE_MTAG_REQUEST_COMPLETE Message

When the application receives a [TXUSER_TRACE_MTAG_REQUEST_COMPLETE \(section 2.2.8.3.5.2\)](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Get Response:
 - Return a success result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.3.5.2 Receiving a TXUSER_TRACE_MTAG_REQUEST_FAILED or TXUSER_TRACE_MTAG_TX_NOT_FOUND Message

When the application receives one of these messages, the application MUST perform the following actions:

- If the connection state is Awaiting Get Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.3.5.3 Connection Disconnected

When a [CONNTYPE_TXUSER_TRACE \(section 2.2.8.3.5\)](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Get Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.5.4 Transaction Manager Administration

3.3.5.4.1 CONNTYPE_TXUSER_GETSECURITYFLAGS as Initiator

For all messages that are received in this connection type, the application MUST process the message as specified in section [3.1](#). The transaction manager MUST also follow the processing rules as specified in the following sections.

3.3.5.4.1.1 Receiving a TXUSER_GETSECURITYFLAGS_MTAG_FETCHED Message

When the application receives a [TXUSER_TRACE_MTAG_REQUEST_COMPLETE \(section 2.2.8.3.5.2\)](#) message, the application MUST perform the following actions:

- If the connection state is Awaiting Get Response:
 - Return a success result and the following message information to the higher-layer business logic:
 - The **grfNetworkDtcAccess** field.
 - The **grfXaTransactions** field.
 - Set the connection state to [Ended \(section 3.2.1.4.14\)](#).
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.3.5.4.1.2 CONNTYPE_TXUSER_GETSECURITYFLAGS Connection Disconnected

When a [CONNTYPE_TXUSER_GETSECURITYFLAGS \(section 2.2.8.4.1\)](#) connection is disconnected, the application MUST perform the following actions:

- If the connection state is Awaiting Get Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.3.6 Timer Events

None.

3.3.7 Other Local Events

None.

3.4 Transaction Manager Communicating with Application Details

3.4.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with the behavior that is described in this document.

The transaction manager communicating with an application facet MUST maintain all the data elements that are specified in section [3.2.1](#).

The transaction manager communicating with an application facet MUST also maintain the following data elements:

- **Associates Table:** A table of entries to lists of connection objects of type [CONNTYPE_TXUSER_ASSOCIATE \(section 2.2.8.2.1.1\)](#), keyed by the identifier of the transaction with which the connections are associated.

Enlistment objects that are created by the transaction manager communicating with an application facet MUST provide the following properties, as specified in section [3.2.1.3](#):

- **Name:** An empty string.

- **Identifier:** An empty string.

The transaction manager communicating with an application facet MUST provide the states in the following sections for its supported connection types. The connection types that a transaction manager communicating with an application facet MUST provide for each supported protocol version are as specified in section [2.2.8](#).

3.4.1.1 CONNTYPE_TXUSER_BEGINNER Acceptor States

The transaction manager communicating with an application MUST act as an acceptor for the [CONNTYPE_TXUSER_BEGINNER](#) connection type. In this role, the transaction manager communicating with an application MUST provide support for the following states:

- Idle
- Beginning Transaction
- Active
- Aborting Transaction
- Transaction Aborted
- Committing Transaction
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_BEGINNER acceptor states.

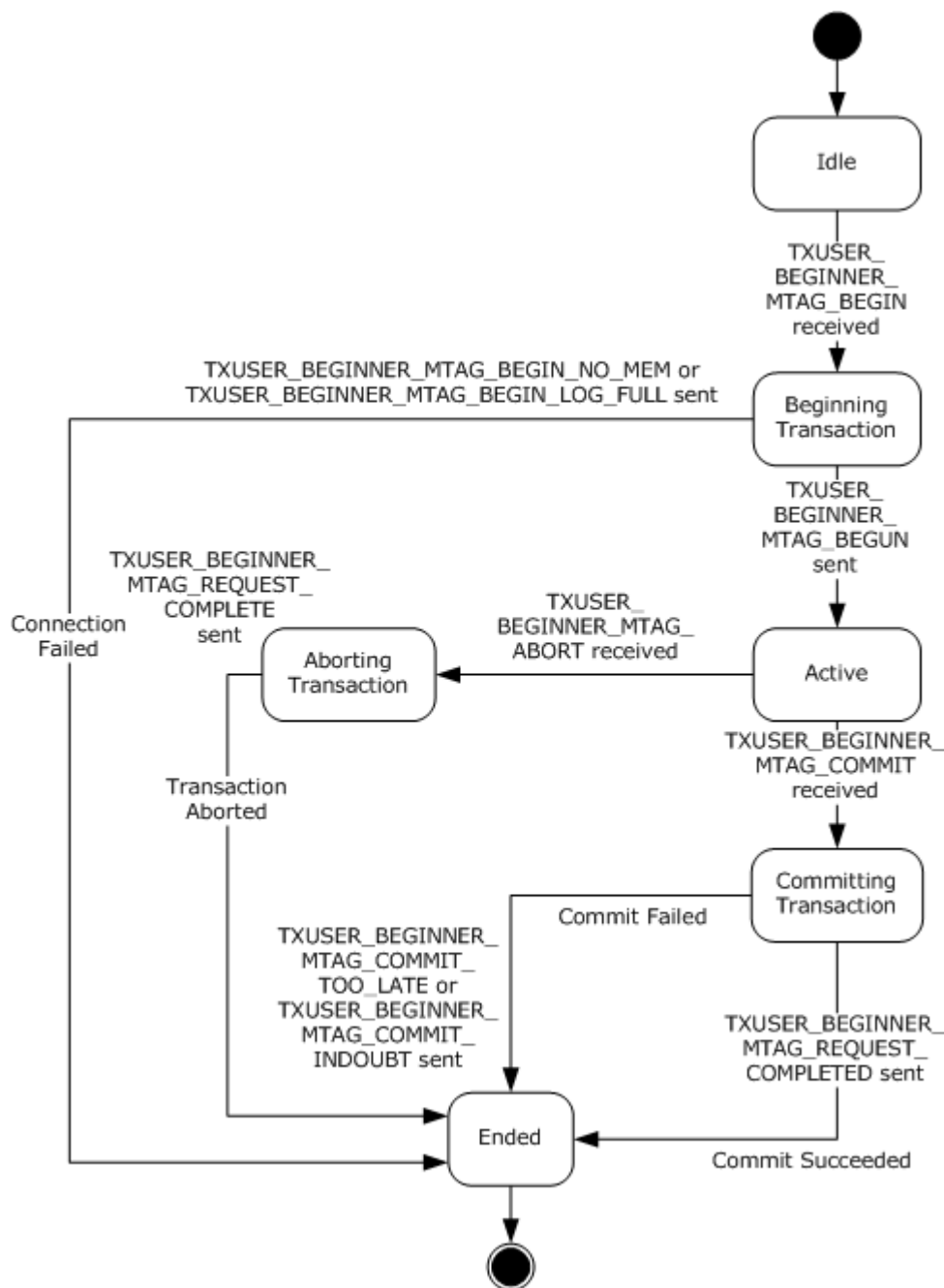


Figure 28: CONNTYPE_TXUSER_BEGINNER Acceptor States

3.4.1.1.1 Idle

The Idle state is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_BEGINNER_MTAG_BEGIN Message \(section 3.4.5.1.1.1\).](#)

3.4.1.1.2 Beginning Transaction

The following events are processed in the Beginning Transaction state:

- [Create Transaction Success \(section 3.4.7.8\).](#)
- [Create Transaction Failure \(section 3.4.7.7\).](#)

3.4.1.1.3 Active

The following events are processed in this state:

- [Receiving a TXUSER BEGINNER MTAG COMMIT Message \(section 3.4.5.1.1.2\).](#)
- [Receiving a TXUSER BEGINNER MTAG ABORT Message \(section 3.4.5.1.1.3\).](#)
- [Unilaterally Aborted \(section 3.4.7.22\).](#)

3.4.1.1.4 Aborting Transaction

The following events are processed in the Aborting Transaction state:

- [Rollback Complete \(section 3.4.7.17\)](#)

3.4.1.1.5 Transaction Aborted

The following events are processed in this state:

- [Receiving a TXUSER BEGINNER MTAG COMMIT Message \(section 3.4.5.1.1.2\).](#)

3.4.1.1.6 Committing Transaction

The following events are processed in this state:

- [Phase One Complete \(section 3.4.7.13\).](#)

3.4.1.1.7 Ended

This is the final state.

3.4.1.2 CONNTYPE_TXUSER_BEGIN2 Acceptor States

The transaction manager communicating with an application MUST act as an acceptor for the [CONNTYPE_TXUSER_BEGIN2](#) connection type. In this role, the transaction manager communicating with an application MUST provide support for the following states:

- Idle
- Beginning Transaction
- Active
- Modifying Timeout
- Aborting Transaction
- Committing Transaction

- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_BEGIN2 acceptor states.

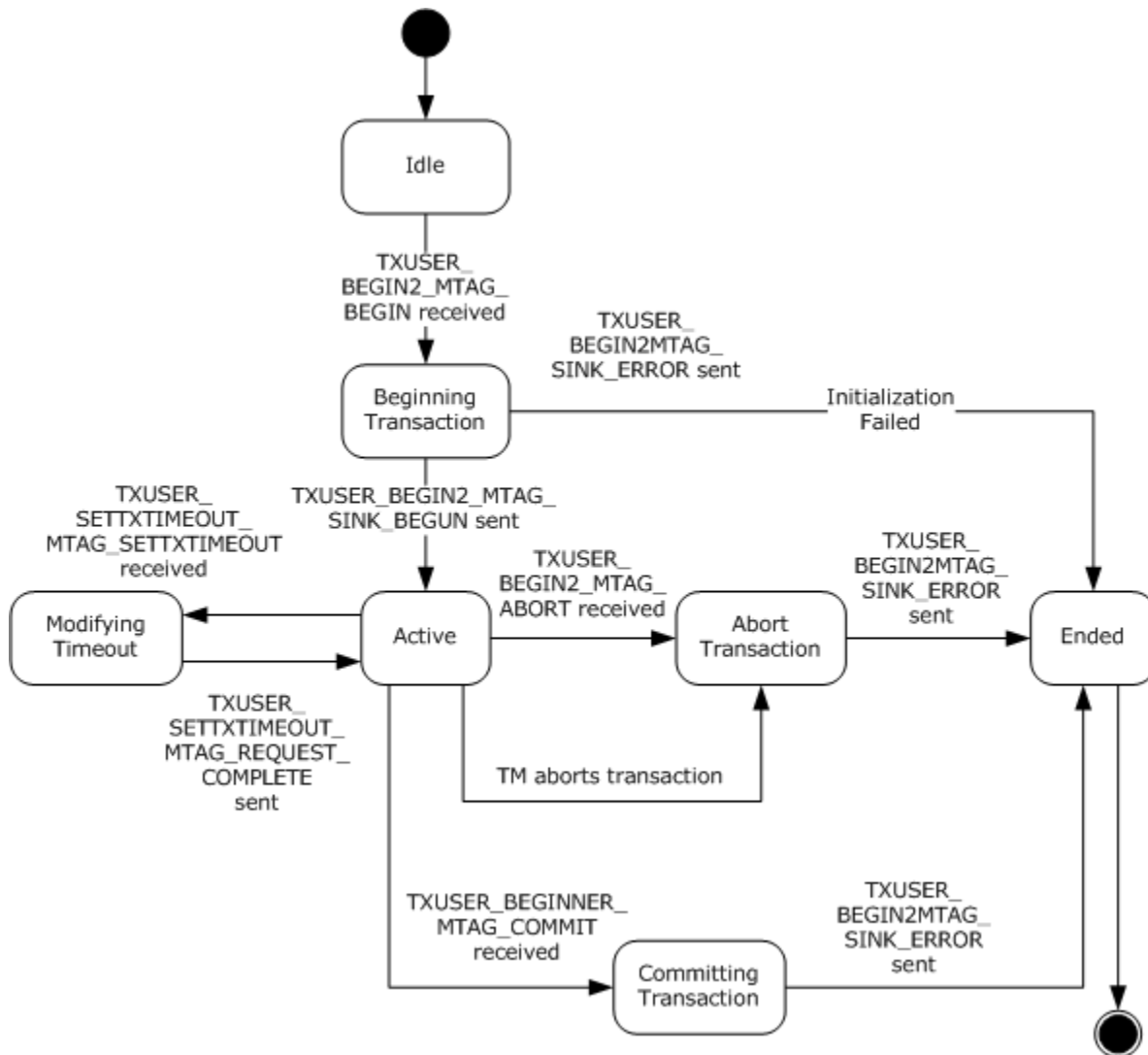


Figure 29: CONNTYPE_TXUSER_BEGIN2 Acceptor States

3.4.1.2.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_BEGIN2_MTAG_BEGIN Message \(section 3.4.5.1.2.1\).](#)

3.4.1.2.2 Beginning Transaction

The following events are processed in this state:

- [Create Transaction Success \(section 3.4.7.8\).](#)
- [Create Transaction Failure \(section 3.4.7.7\).](#)

3.4.1.2.3 Active

The following events are processed in this state:

- [Receiving a TXUSER SETTXTIMEOUT MTAG SETTXTIMEOUT Message \(section 3.4.5.1.2.2\).](#)
- [Receiving a TXUSER BEGIN2 MTAG COMMIT Message \(section 3.4.5.1.2.3\).](#)
- [Receiving a TXUSER BEGIN2 MTAG ABORT Message \(section 3.4.5.1.2.4\).](#)
- [Unilaterally Aborted \(section 3.4.7.22\).](#)

3.4.1.2.4 Modifying Timeout

The following events are processed in this state:

- [Set Transaction Timeout Success \(section 3.4.7.21\).](#)
- [Set Transaction Timeout Failure \(section 3.4.7.20\).](#)

3.4.1.2.5 Aborting Transaction

The following events are processed in this state:

- [Rollback Complete \(section 3.4.7.17\).](#)

3.4.1.2.6 Committing Transaction

The following events are processed in this state:

- [Phase One Complete \(section 3.4.7.13\).](#)

3.4.1.2.7 Ended

This is the final state.

3.4.1.3 CONNTYPE_TXUSER_PROMOTE Acceptor States

The transaction manager communicating with an application MUST act as an acceptor for the [CONNTYPE_TXUSER_PROMOTE](#) connection type. In this role, the transaction manager communicating with an application MUST provide support for the following states:

- Idle
- Beginning Transaction
- Active
- Modifying Timeout
- Aborting Transaction
- Committing Transaction

- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_PROMOTE acceptor states.

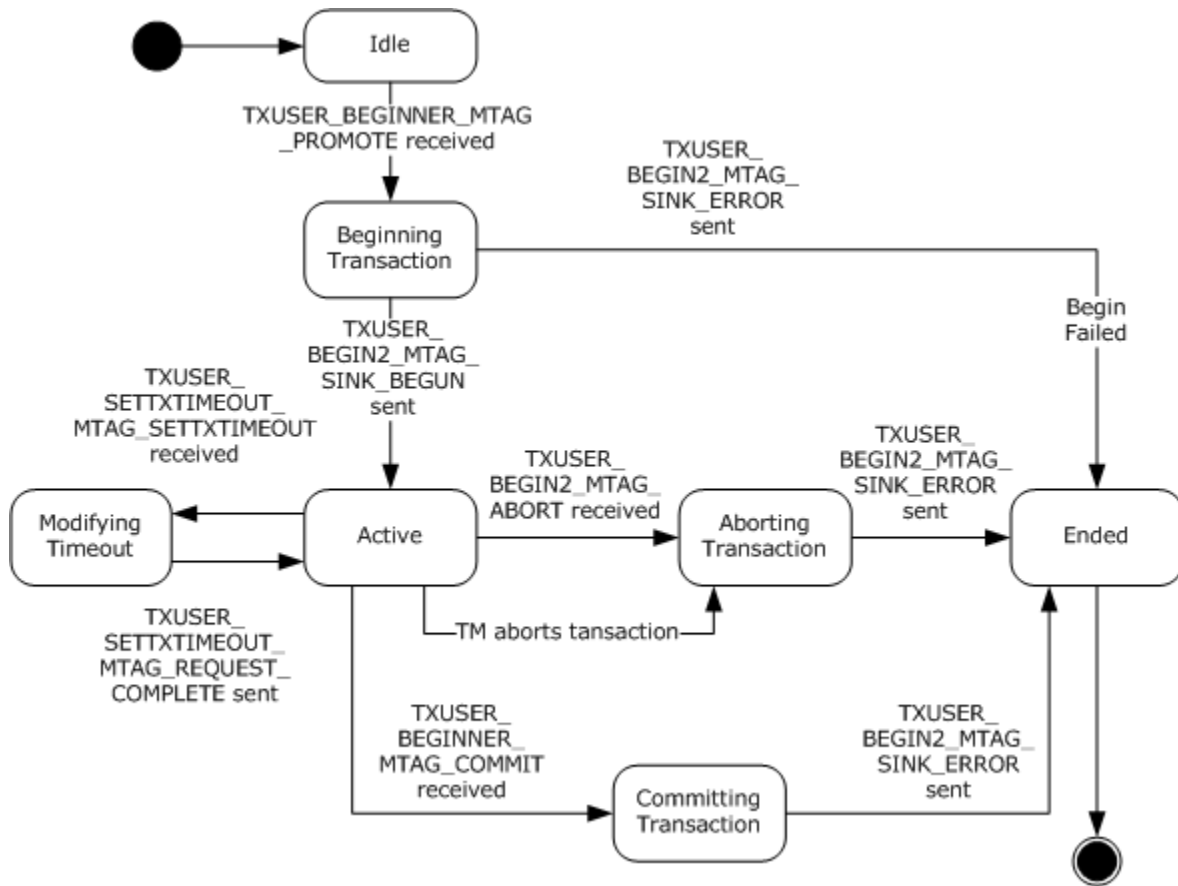


Figure 30: CONNTYPE_TXUSER_PROMOTE Acceptor States

3.4.1.3.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_BEGINNER_MTAG_PROMOTE Message \(section 3.4.5.1.3.1\).](#)

3.4.1.3.2 Beginning Transaction

The following events are processed in this state:

- [Create Transaction Success \(section 3.4.7.8\).](#)
- [Create Transaction Failure \(section 3.4.7.7\).](#)

3.4.1.3.3 Active

The following events are processed in this state:

- [Receiving a TXUSER SETTXTIMEOUT MTAG SETTXTIMEOUT Message \(section 3.4.5.1.2.2\).](#)
- [Receiving a TXUSER BEGIN2 MTAG COMMIT Message \(section 3.4.5.1.2.3\).](#)
- [Receiving a TXUSER BEGIN2 MTAG ABORT Message \(section 3.4.5.1.2.4\).](#)
- [Receiving a TXUSER BEGINNER MTAG PROMOTE Message \(section 3.4.5.1.3.1\).](#)
- [Unilaterally Aborted \(section 3.4.7.22\).](#)

3.4.1.3.4 Modifying Timeout

The following events are processed in this state:

- [Set Transaction Timeout Success \(section 3.4.7.21\).](#)
- [Set Transaction Timeout Failure \(section 3.4.7.20\).](#)

3.4.1.3.5 Aborting Transaction

The following events are processed in this state:

- [Rollback Complete \(section 3.4.7.17\).](#)

3.4.1.3.6 Committing Transaction

The following events are processed in this state:

- [Phase One Complete \(section 3.4.7.13\).](#)

3.4.1.3.7 Ended

This is the final state.

3.4.1.4 CONNTYPE_TXUSER_ASSOCIATE Acceptor States

The transaction manager communicating with an application MUST act as an acceptor for the [CONNTYPE_TXUSER_ASSOCIATE](#) connection type. In this role, the transaction manager communicating with an application MUST provide support for the following states:

- Idle
- Processing Associate Request
- Active
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_ASSOCIATE acceptor states.

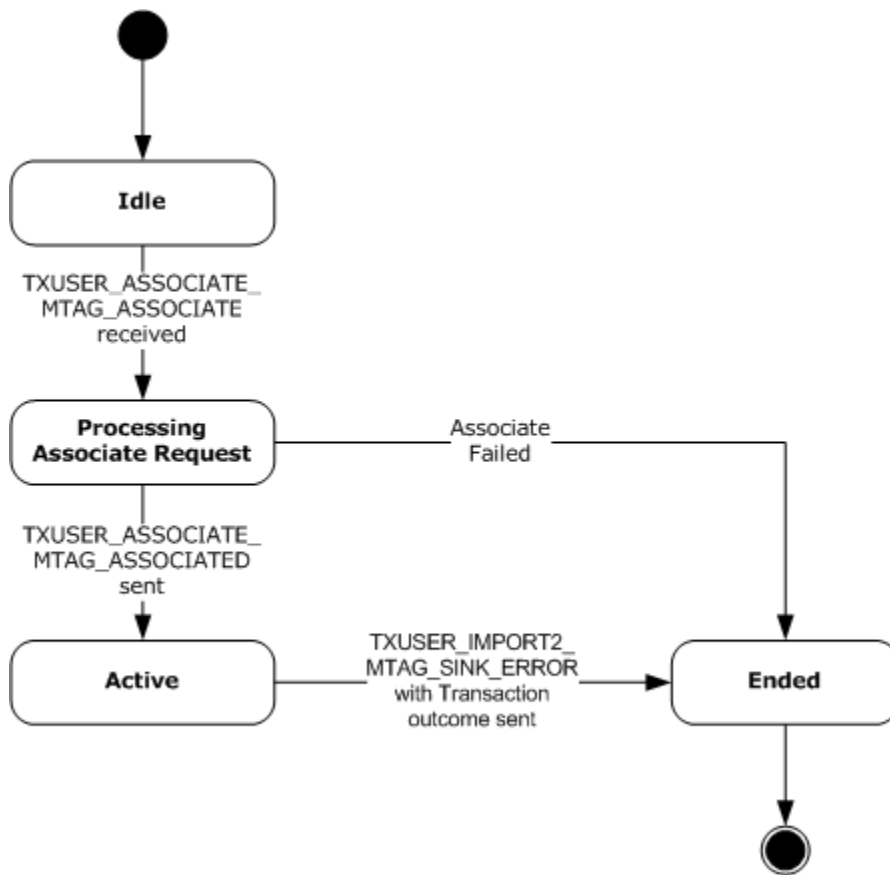


Figure 31: CONNTYPE_TXUSER_ASSOCIATE Acceptor States

3.4.1.4.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_ASSOCIATE_MTAG_ASSOCIATE Message \(section 3.4.5.2.1.1.1\).](#)

3.4.1.4.2 Processing Associate Request

The following events are processed in this state:

- [Associate Transaction Success \(section 3.4.7.2\).](#)
- [Associate Transaction Failure \(section 3.4.7.1\).](#) This event applies to these messages:
 - [TXUSER_ASSOCIATE_MTAG_COMM_FAILED \(section 2.2.8.2.1.1.3\)](#)
 - [TXUSER_ASSOCIATE_MTAG_LOG_FULL_LOCAL \(section 2.2.8.2.1.1.5\)](#)
 - [TXUSER_ASSOCIATE_MTAG_LOG_FULL_REMOTE \(section 2.2.8.2.1.1.6\)](#)
 - [TXUSER_ASSOCIATE_MTAG_NO_MEM_LOCAL \(section 2.2.8.2.1.1.7\)](#)
 - [TXUSER_ASSOCIATE_MTAG_NO_MEM_REMOTE \(section 2.2.8.2.1.1.8\)](#)

- [TXUSER ASSOCIATE MTAG TOO LATE \(section 2.2.8.2.1.1.9\)](#)
- [TXUSER ASSOCIATE MTAG TOO MANY REMOTE \(section 2.2.8.2.1.1.11\)](#)
- [TXUSER ASSOCIATE MTAG TX NOT FOUND \(section 2.2.8.2.1.1.12\)](#)

3.4.1.4.3 Active

The following events are processed in this state:

- Begin Voting (section [3.4.7.6](#)).
- Begin Commit (section [3.4.7.3](#)).
- Begin Rollback (section [3.4.7.5](#)).
- Begin In Doubt (section [3.4.7.4](#)).

3.4.1.4.4 Ended

This is the final state.

3.4.1.5 CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS Acceptor States

The transaction manager communicating with an application MUST act as an acceptor for the [CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS](#) connection type. In this role, the transaction manager communicating with an application MUST provide support for the following states:

- Idle
- Processing Inquiry
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS acceptor states.

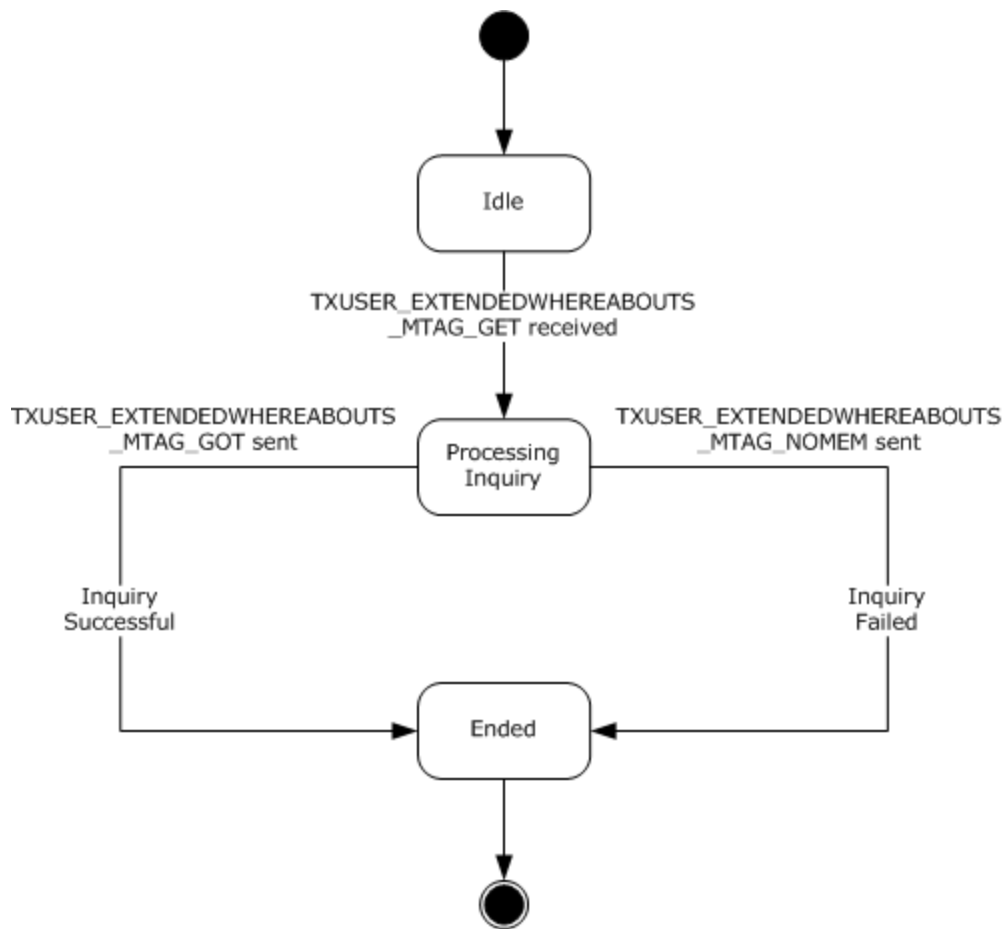


Figure 32: CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS Acceptor States

3.4.1.5.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_EXTENDEDWHEREABOUTS_MTAG_GET Message \(section 3.4.5.2.1.1\).](#)

3.4.1.5.2 Processing Inquiry

This is a transient state that is assumed during the synchronous processing of a request. No events are processed in this state.

3.4.1.5.3 Ended

This is the final state.

3.4.1.6 CONNTYPE_TXUSER_IMPORT Acceptor States

The transaction manager communicating with an application **MUST** act as an acceptor for the [CONNTYPE_TXUSER_IMPORT](#) connection type. In this role, the transaction manager communicating with an application **MUST** provide support for the following states:

- Idle
- Processing Import Request
- Active
- Too Late to Abort
- Processing Abort Request
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_IMPORT acceptor states.

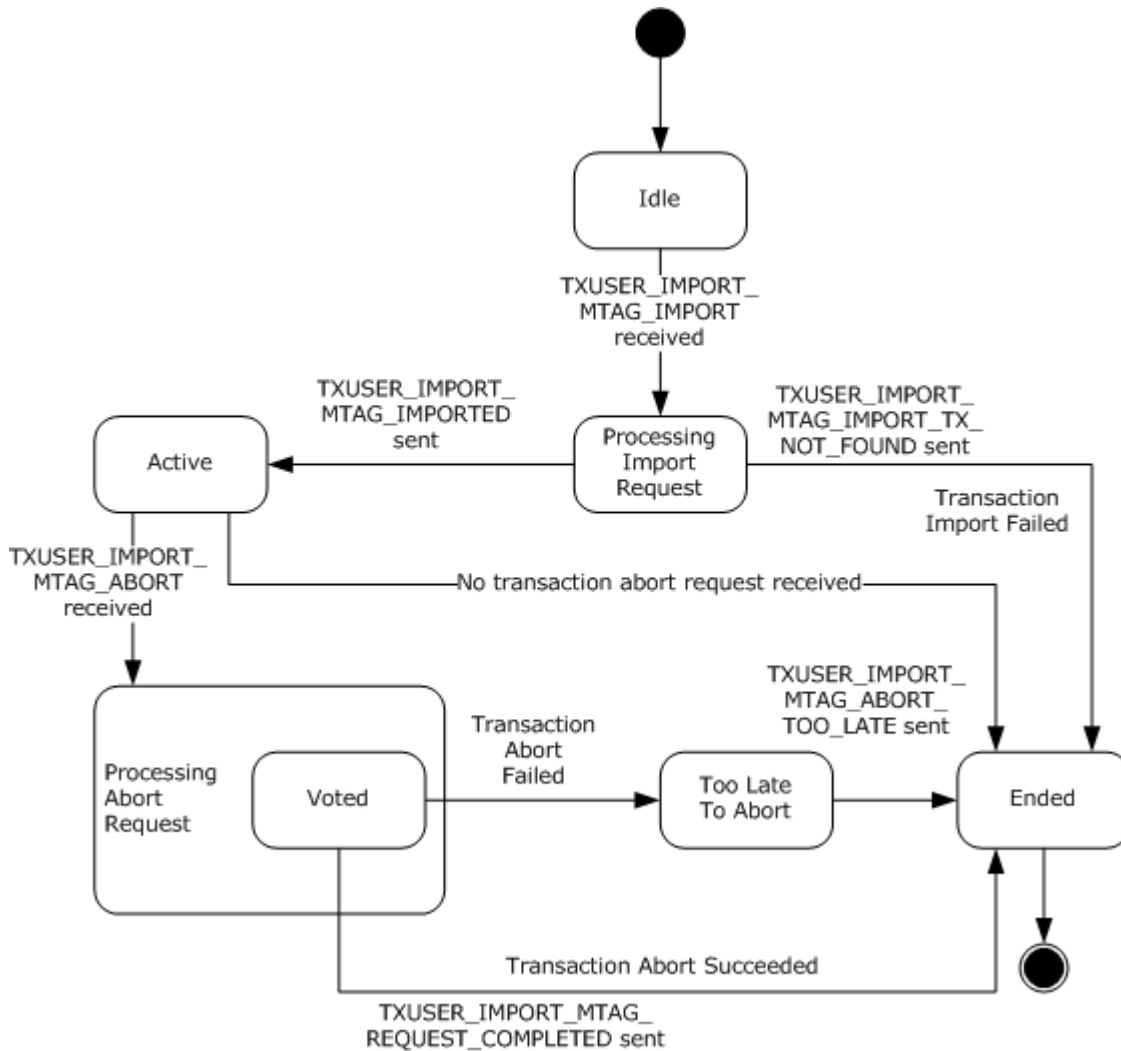


Figure 33: CONNTYPE_TXUSER_IMPORT Acceptor States

3.4.1.6.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_IMPORT_MTAG_IMPORT Message \(section 3.4.5.2.2.3.1\).](#)

3.4.1.6.2 Processing Import Request

The following events are processed in this state:

- [Create Voter Enlistment Success \(section 3.4.7.10\).](#)
- [Create Voter Enlistment Failure \(section 3.4.7.9\).](#)

3.4.1.6.3 Active

The following events are processed in this state:

- [Receiving a TXUSER_IMPORT_MTAG_ABORT Message \(section 3.4.5.2.2.3.2\).](#)
- [Begin Voting \(section 3.4.7.6\).](#)
- [Begin Commit \(section 3.4.7.3\).](#)
- [Begin Rollback \(section 3.4.7.5\).](#)
- [Begin In Doubt \(section 3.4.7.4\).](#)

3.4.1.6.4 Too Late to Abort

The following events are processed in this state:

- [Receiving a TXUSER_IMPORT_MTAG_ABORT Message \(section 3.4.5.2.2.3.2\).](#)
- [Begin Rollback \(section 3.4.7.5\).](#)
- [Begin Commit \(section 3.4.7.3\).](#)
- [Begin In Doubt \(section 3.4.7.4\).](#)

3.4.1.6.5 Processing Abort Request

This is a transient state that is assumed during the synchronous processing of a request to abort a transaction. No events are processed in this state.

3.4.1.6.6 Ended

This is the final state.

3.4.1.7 CONNTYPE_TXUSER_IMPORT2 Acceptor States

The transaction manager communicating with an application MUST act as an acceptor for the [CONNTYPE_TXUSER_IMPORT2](#) connection type. In this role, the transaction manager communicating with an application MUST provide support for the following states:

- Idle

- Processing Import Request
- Active
- Too Late to Abort
- Processing Abort Request
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_IMPORT2 acceptor states.

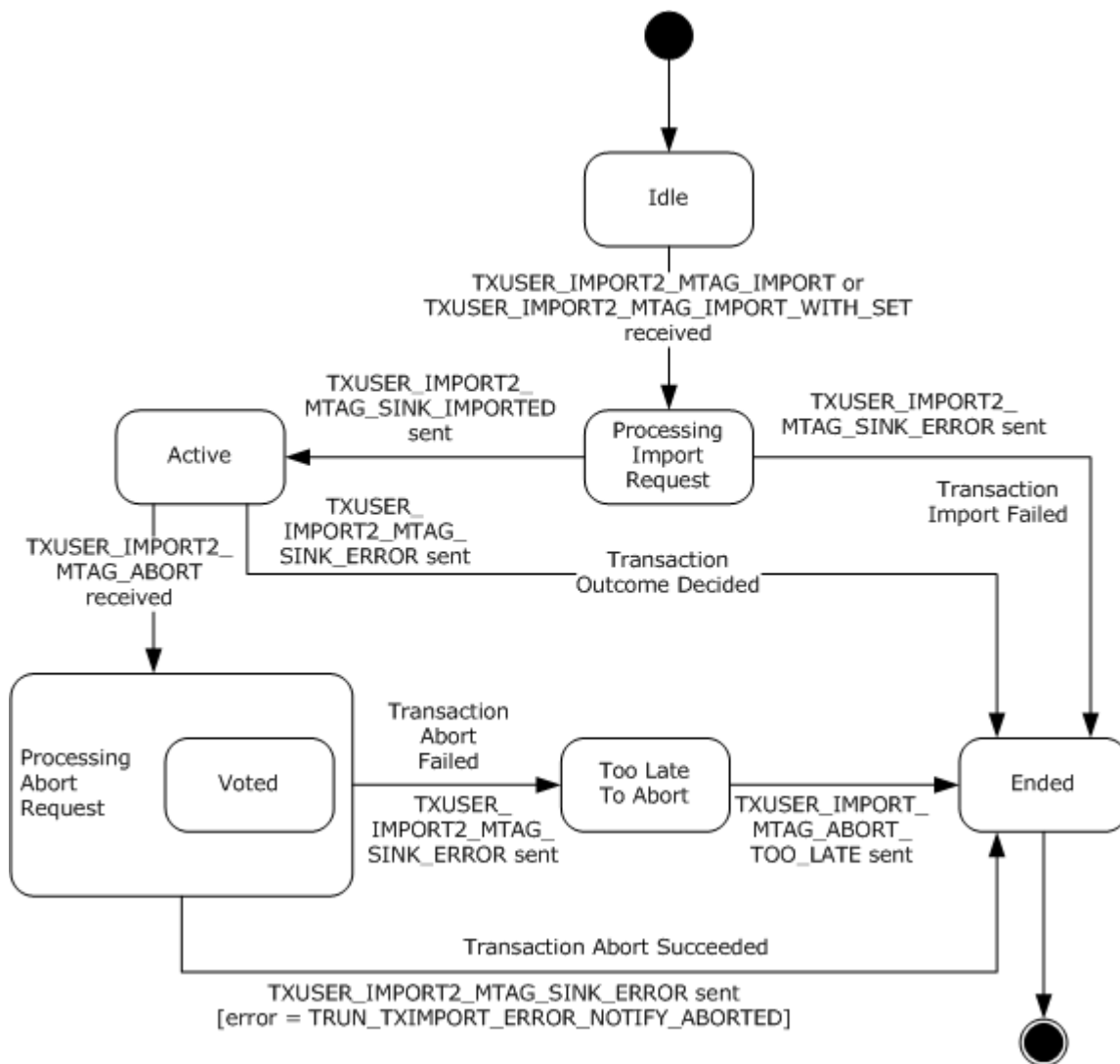


Figure 34: CONNTYPE_TXUSER_IMPORT2 Acceptor States

3.4.1.7.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_IMPORT2_MTAG_IMPORT Message \(section 3.4.5.2.2.4.1\).](#)
- [Receiving a TXUSER_IMPORT2_MTAG_IMPORT_WITH_SET Message \(section 3.4.5.2.2.4.2\).](#)

3.4.1.7.2 Processing Import Request

The following events are processed in this state:

- [Set Transaction Attributes Success \(section 3.4.7.19\).](#)
- [Set Transaction Attributes Failure \(section 3.4.7.18\).](#)
- [Create Voter Enlistment Success \(section 3.4.7.10\).](#)
- [Create Voter Enlistment Failure \(section 3.4.7.9\).](#)

3.4.1.7.3 Active

The following events are processed in this state:

- [Receiving a TXUSER_IMPORT2_MTAG_ABORT message \(section 3.4.5.2.2.4.3\).](#)
- [Begin Voting \(section 3.4.7.6\).](#)
- [Begin Commit \(section 3.4.7.3\).](#)
- [Begin Rollback \(section 3.4.7.5\).](#)
- [Begin In Doubt \(section 3.4.7.4\).](#)

3.4.1.7.4 Too Late to Abort

The following events are processed in this state:

- [Receiving a TXUSER_IMPORT2_MTAG_ABORT message \(section 3.4.5.2.2.4.3\).](#)
- [Begin Commit \(section 3.4.7.3\).](#)
- [Begin Rollback \(section 3.4.7.5\).](#)
- [Begin In Doubt \(section 3.4.7.4\).](#)

3.4.1.7.5 Processing Abort Request

This is a transient state that is assumed during the synchronous processing of a request to abort a transaction. No events are processed in this state.

3.4.1.7.6 Ended

This is the final state.

3.4.1.8 CONNTYPE_TXUSER_EXPORT Acceptor States

The transaction manager communicating with an application **MUST** act as an acceptor for the [CONNTYPE_TXUSER_EXPORT](#) connection type. In this role, the transaction manager communicating with an application **MUST** provide support for the following states:

- Idle
- Processing Connection Request
- Connection Active
- Processing Push Operation Request
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_EXPORT acceptor states.

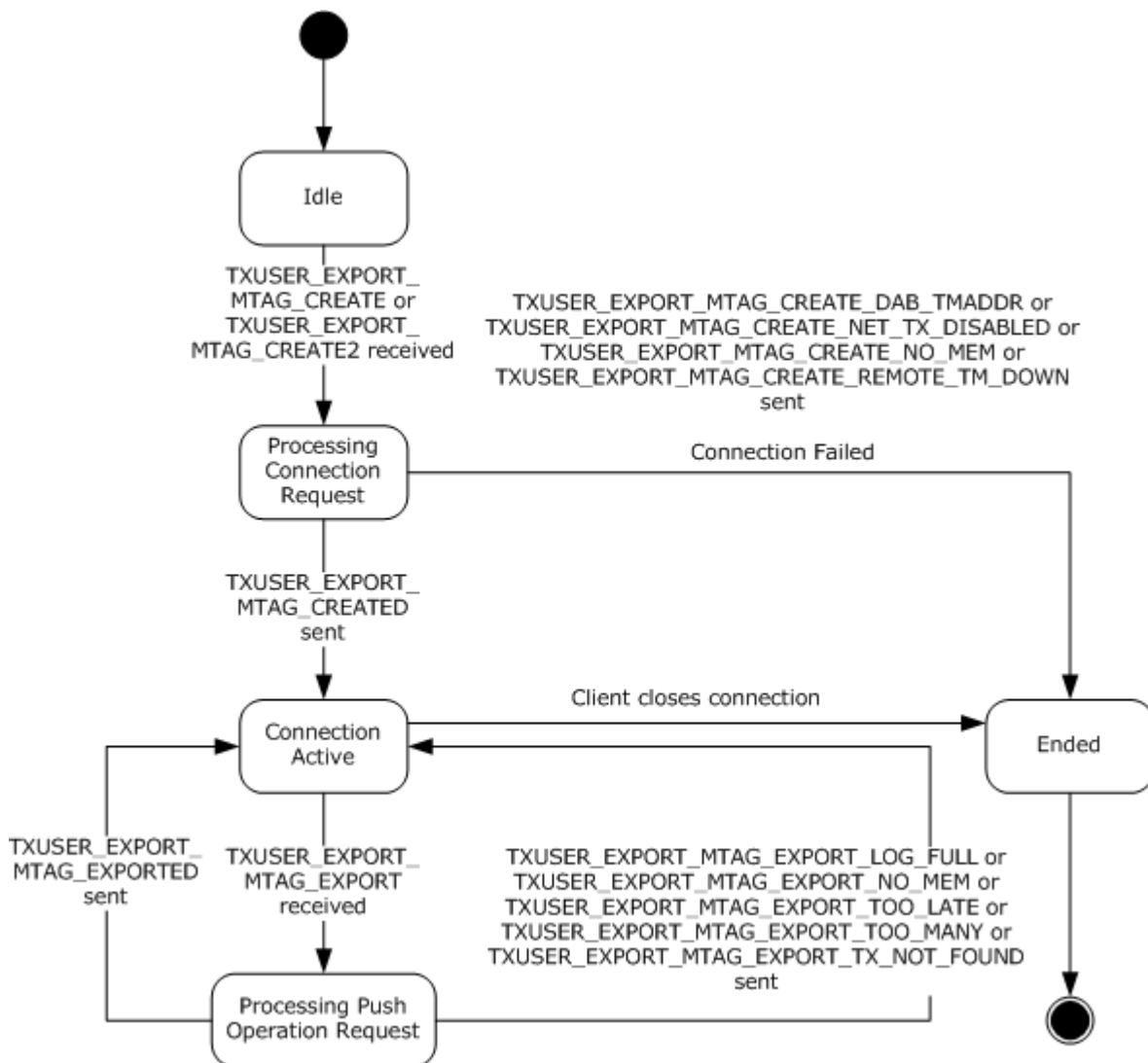


Figure 35: CONNTYPE_TXUSER_EXPORT Acceptor States

3.4.1.8.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_EXPORT_MTAG_CREATE Message \(section 3.4.5.2.2.2.1\).](#)
- [Receiving a TXUSER_EXPORT_MTAG_CREATE2 Message \(section 3.4.5.2.2.2.2\).](#)

3.4.1.8.2 Processing Connection Request

This is a transient state that is assumed during the synchronous processing of a create export request. No events are processed in this state.

3.4.1.8.3 Connection Active

The following events are processed in this state:

- [Receiving a TXUSER_EXPORT_MTAG_EXPORT Message \(section 3.4.5.2.2.2.3\).](#)

3.4.1.8.4 Processing Push Operation Request

The following events are processed in this state:

- [Export Transaction Success \(section 3.4.7.12\).](#)
- [Export Transaction Failure \(section 3.4.7.11\).](#)

3.4.1.8.5 Ended

This is the final state.

3.4.1.9 CONNTYPE_TXUSER_GETTXDETAILS Acceptor States

The transaction manager communicating with an application MUST act as an acceptor for the [CONNTYPE_TXUSER_GETTXDETAILS](#) connection type. In this role, the transaction manager communicating with an application MUST provide support for the following states:

- Idle
- Processing Inquiry
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_GETTXDETAILS acceptor states.

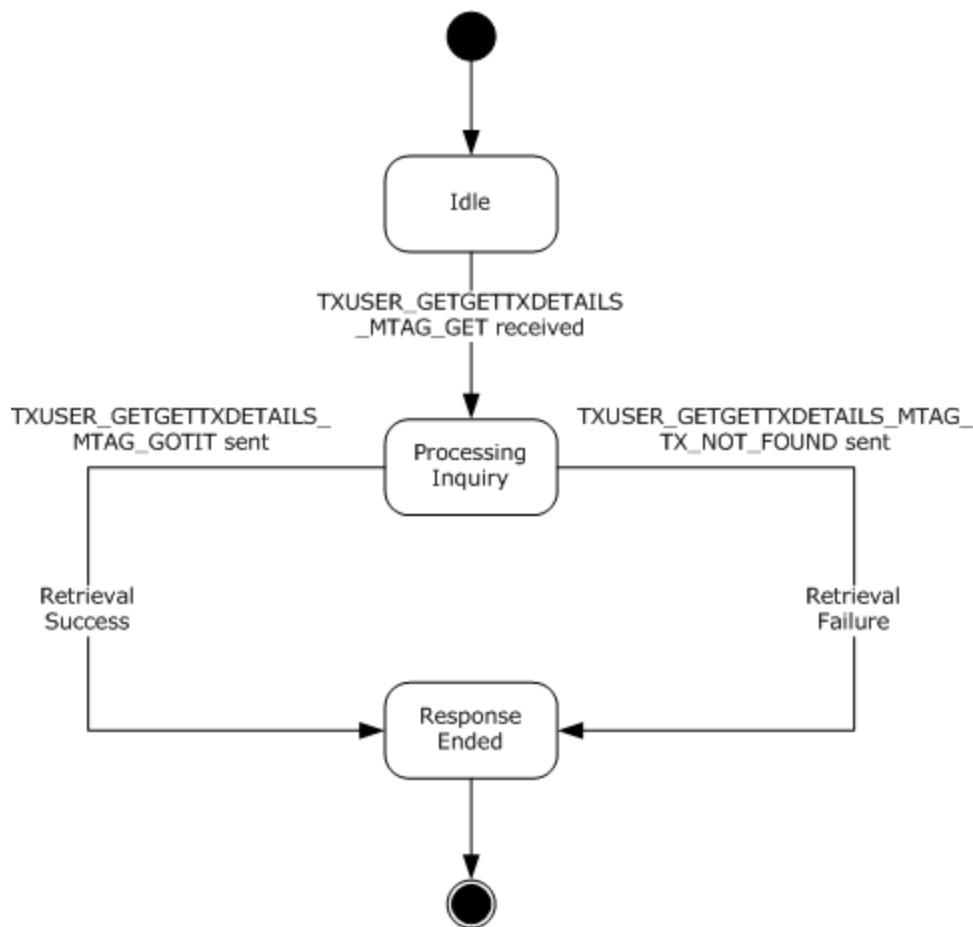


Figure 36: CONNTYPE_TXUSER_GETTXDETAILS Acceptor States

3.4.1.9.1 Idle

This is the initial State. The following events are processed in this state:

- [Receiving a TXUSER_GETTXDETAILS_MTAG_GET Message \(section 3.4.5.3.1.1\).](#)

3.4.1.9.2 Processing Inquiry

This is a transient state that is assumed during the synchronous processing of a request for the transaction details. No events are processed in this state.

3.4.1.9.3 Ended

This is the final state.

3.4.1.10 CONNTYPE_TXUSER_RESOLVE Acceptor States

The transaction manager communicating with an application **MUST** act as an acceptor for the [CONNTYPE_TXUSER_RESOLVE](#) connection type. In this role, the transaction manager communicating with an application **MUST** provide support for the following states:

- Idle
- Processing Abort Response
- Processing Forget Response
- Processing Commit Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_RESOLVE acceptor states.

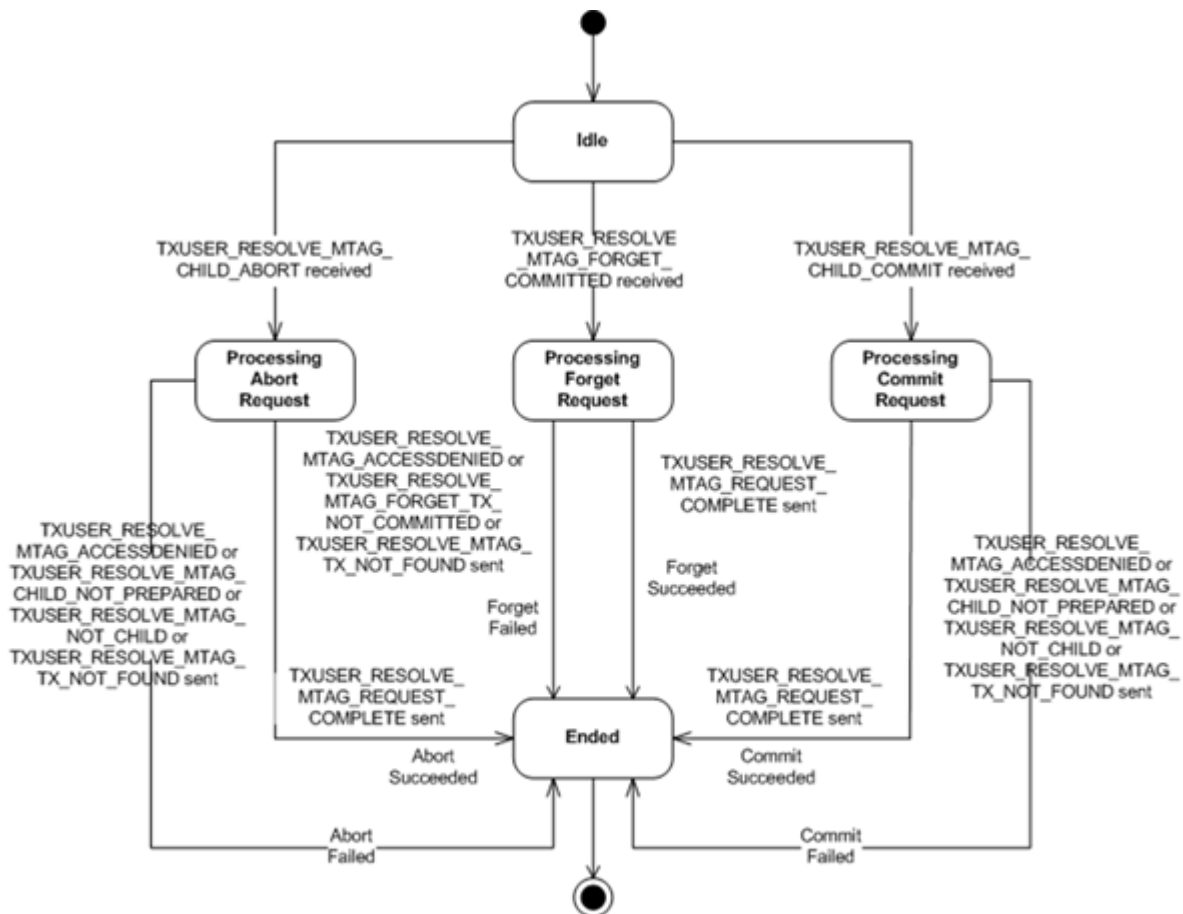


Figure 37: CONNTYPE_TXUSER_RESOLVE Acceptor States

3.4.1.10.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_RESOLVE_MTAG_CHILD_ABORT Message \(section 3.4.5.3.2.1\).](#)

- [Receiving a TXUSER_RESOLVE_MTAG_CHILD_COMMIT Message \(section 3.4.5.3.2.2\).](#)
- [Receiving a TXUSER_RESOLVE_MTAG_FORGET_COMMITTED Message \(section 3.4.5.3.2.3\).](#)

3.4.1.10.2 Processing Abort Request

The following events are processed in this state:

- [Resolve Transaction Complete \(section 3.4.7.16\).](#)

3.4.1.10.3 Processing Forget Request

The following events are processed in this state:

- [Resolve Transaction Complete \(section 3.4.7.16\).](#)

3.4.1.10.4 Processing Commit Request

The following events are processed in this state:

- [Resolve Transaction Complete \(section 3.4.7.16\).](#)

3.4.1.10.5 Ended

This is the final state.

3.4.1.11 CONNTYPE_TXUSER_SETTXTIMEOUT Acceptor States

The transaction manager communicating with an application MUST act as an acceptor for the [CONNTYPE_TXUSER_SETTXTIMEOUT](#) connection type. In this role, the transaction manager communicating with an application MUST provide support for the following states:

- Idle
- Resetting Timeout
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_SETTXTIMEOUT acceptor states.

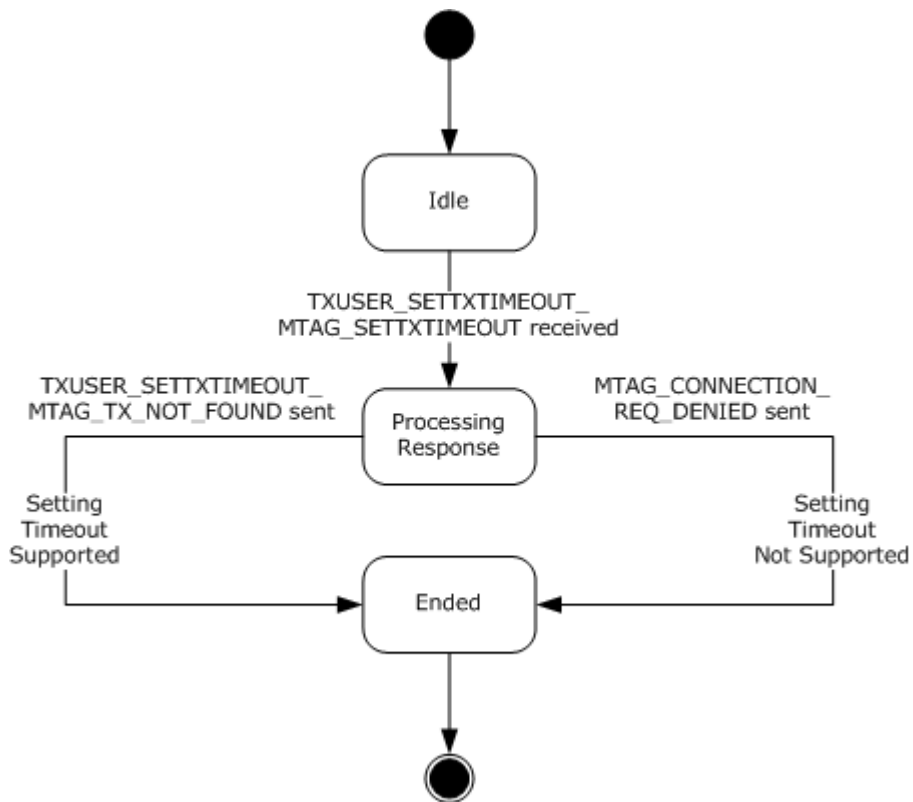


Figure 38: CONNTYPE_TXUSER_SETTXXTIMEOUT Acceptor States

3.4.1.11.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_SETTXXTIMEOUT_MTAG_SETTXXTIMEOUT Message \(section 3.4.5.3.3.1\).](#)

3.4.1.11.2 Processing Request

The following events are processed in this state:

- [Set Transaction Timeout Success \(section 3.4.7.21\).](#)
- [Set Transaction Timeout Failure \(section 3.4.7.20\).](#)

3.4.1.11.3 Ended

This is the final state.

3.4.1.12 CONNTYPE_TXUSER_SETTXXTIMEOUT2 Acceptor States

The transaction manager communicating with an application **MUST** act as an acceptor for the [CONNTYPE_TXUSER_SETTXXTIMEOUT2](#) connection type. In this role, the transaction manager communicating with an application **MUST** provide support for the following states:

- Idle

- Processing Request
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_SETTXTIMEOUT2 acceptor states.

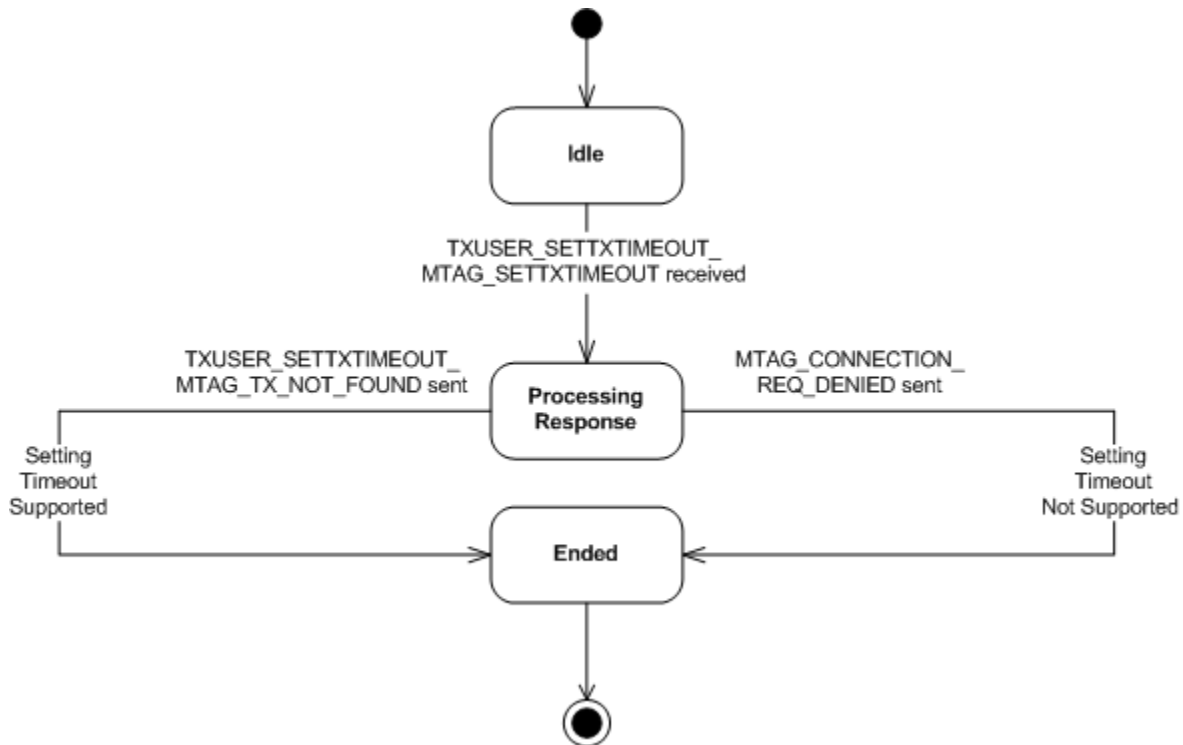


Figure 39: CONNTYPE_TXUSER_SETTXTIMEOUT2 Acceptor States

3.4.1.12.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT Message \(section 3.4.5.1.2.2\).](#)

3.4.1.12.2 Processing Request

This is a transient state that is assumed during the synchronous processing of a request to set a transaction time-out. No events are processed in this state.

3.4.1.12.3 Ended

This is the final state.

3.4.1.13 CONNTYPE_TXUSER_TRACE Acceptor States

The transaction manager communicating with an application MUST act as an acceptor for the [CONNTYPE_TXUSER_TRACE](#) connection type. In this role, the transaction manager communicating with an application MUST provide support for the following states:

- Idle
- Processing Trace Response
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_TRACE acceptor states.

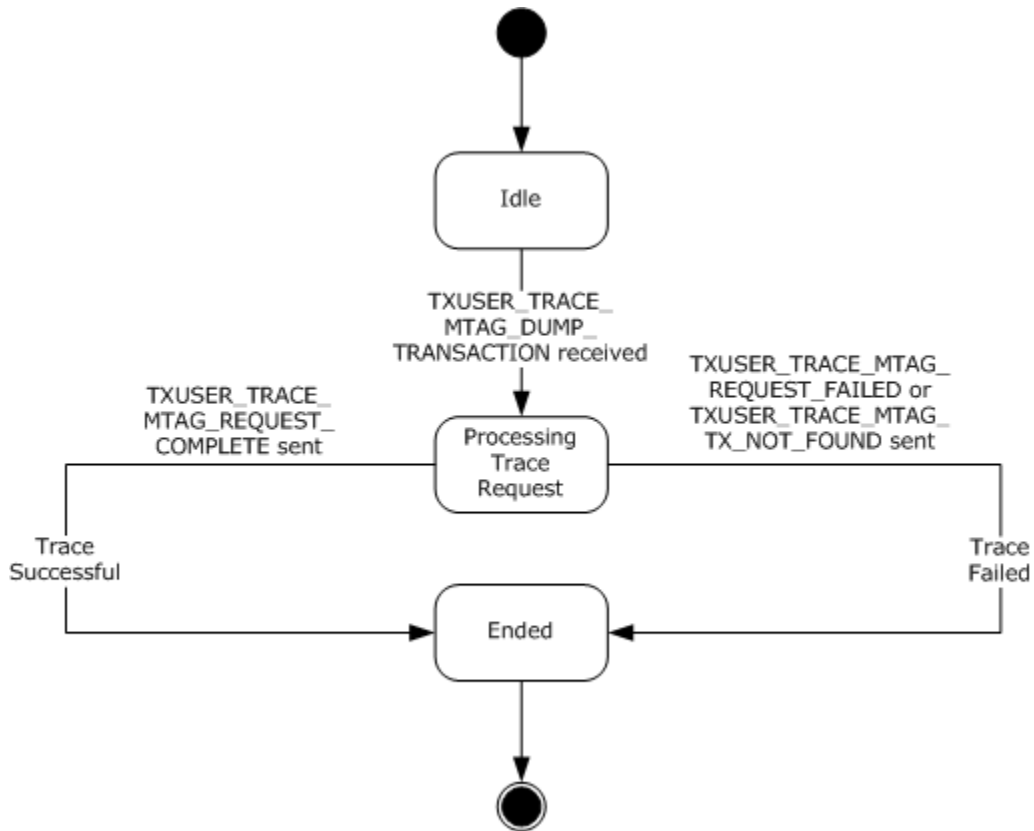


Figure 40: CONNTYPE_TXUSER_TRACE Acceptor States

3.4.1.13.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_TRACE_MTAG_DUMP_TRANSACTION Message \(section 3.4.5.3.5.1\).](#)

3.4.1.13.2 Processing Trace Request

This is a transient state that is assumed during the synchronous processing of a request to set a transaction time-out. No events are processed in this state.

3.4.1.13.3 Ended

This is the final state.

3.4.1.14 CONNTYPE_TXUSER_GETSECURITYFLAGS Acceptor States

The transaction manager communicating with an application MUST act as an acceptor for the [CONNTYPE_TXUSER_GETSECURITYFLAGS](#) connection type. In this role, the transaction manager communicating with an application MUST provide support for the following states:

- Idle
- Processing Request
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_GETSECURITYFLAGS acceptor states.

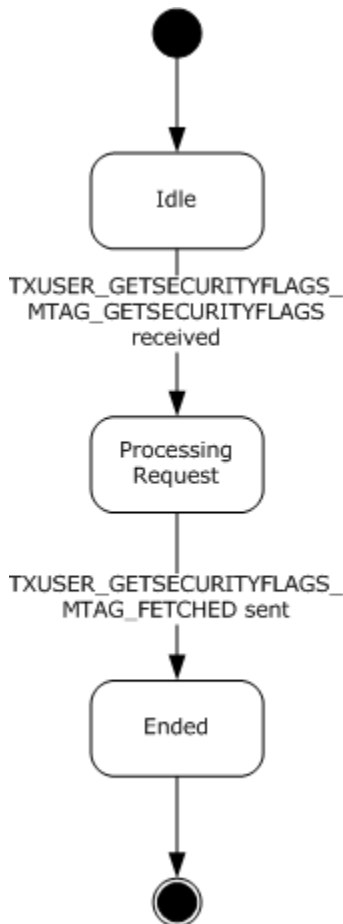


Figure 41: CONNTYPE_TXUSER_GETSECURITYFLAGS Acceptor States

3.4.1.14.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_GETSECURITYFLAGS_MTAG_GETSECURITYFLAGS Message \(section 3.4.5.4.1.1\).](#)

3.4.1.14.2 Processing Request

This is a transient state that is assumed during the synchronous processing of a request to get the security flags. No events are processed in this state.

3.4.1.14.3 Ended

This is the final state.

3.4.2 Timers

No timers apply here.

3.4.3 Initialization

When the transaction manager communicating with an application facet is initialized:

- The transaction manager communicating with an application facet MUST examine the following security flags on the [Core Transaction Manager Facet](#) and perform the following actions:
 - If the Allow Network Access flag is set to false:
 - The transaction manager communicating with an applicationfacet MUST refuse to accept incoming connections from remote machines for all the connection types listed in [3.4.1](#).
 - Otherwise:
 - If the Allow Remote Clients flag is set to false:
 - The transaction manager communicating with an applicationfacet MUST refuse to accept incoming connections from remote machines for the following connection types:
 - [CONNTYPE_TXUSER_ASSOCIATE](#)
 - [CONNTYPE_TXUSER_BEGINNER](#)
 - [CONNTYPE_TXUSER_BEGIN2](#)
 - [CONNTYPE_TXUSER_EXPORT](#)
 - [CONNTYPE_TXUSER_IMPORT](#)
 - [CONNTYPE_TXUSER_IMPORT2](#)
 - [CONNTYPE_TXUSER_PROMOTE](#)
 - If Allow Remote Administration flag is set to false:
 - The transaction manager communicating with an applicationfacet MUST refuse to accept incoming connections from remote machines for the following connection types:
 - [CONNTYPE_TXUSER_GETTXDETAILS](#)
 - [CONNTYPE_TXUSER_RESOLVE](#)
 - [CONNTYPE_TXUSER_TRACE](#)

3.4.4 Higher-Layer Triggered Events

No higher-layer triggered events apply here.

3.4.5 Message Processing Events and Sequencing Rules

3.4.5.1 Transaction Initiation and Completion

3.4.5.1.1 CONNTYPE_TXUSER_BEGINNER as Acceptor

For all messages that are received in this connection type, the transaction manager communicating with an application facet **MUST** process the message as specified in section [3.1](#). The transaction manager communicating with an application facet **MUST** also follow the processing rules that are specified in the following sections.

3.4.5.1.1.1 Receiving a TXUSER_BEGINNER_MTAG_BEGIN Message

When the transaction manager communicating with an application facet receives a [TXUSER_BEGINNER_MTAG_BEGIN \(section 2.2.8.1.1.2\)](#) message, the transaction manager communicating with an application facet **MUST** perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Beginning Transaction.
 - If the transaction manager does not have sufficient memory available to process the message:
 - Send a [TXUSER_BEGINNER_MTAG_BEGIN_NO_MEM \(section 2.2.8.1.1.4\)](#) message using the connection.
 - Set the connection state to [Ended](#).
 - Otherwise:
 - Create a transaction object using the transaction settings provided in the message:
 - Use the **isoLevel** field as the Isolation Level value of the transaction.
 - Use the **dwTimeout** field as the Timeout value of the transaction.
 - Use the **szDesc** field as the Description value of the transaction.
 - Use the **isoFlags** field as the Isolation Flags value of the transaction.
 - Create a new GUID as specified in [\[RFC4122\]](#) and assign it to the **Identifier** field of the transaction object.
 - Add the connection to the connection list of the transaction.
 - Set the **Transaction** field of the connection to the transaction object.
 - Create a new enlistment object with the following values:
 - The transaction manager communicating with an applicationfacet.
 - The transaction object.

- The connection.
- Signal the [Create Transaction \(section 3.2.7.13\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the enlistment object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.1.1.2 Receiving a TXUSER_BEGINNER_MTAG_COMMIT Message

When the transaction manager communicating with an application facet receives a [TXUSER BEGINNER MTAG COMMIT](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Active:
 - Set the connection state to Committing Transaction.
 - Signal the [Begin Phase Zero \(section 3.6.7.4\)](#) event on the [Core Transaction Manager Facet](#) with the transaction object referenced by the transaction object referenced by this connection.
- If the connection state is Transaction Aborted:
 - Send a [TXUSER BEGINNER MTAG COMMIT TOO LATE](#) message using the connection.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.1.1.3 Receiving a TXUSER_BEGINNER_MTAG_ABORT Message

When the transaction manager communicating with an application facet receives a [TXUSER BEGINNER MTAG ABORT](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Active:
 - Set the connection state to Aborting Transaction.
 - Signal the [Begin Rollback \(section 3.4.7.5\)](#) event on the [Core Transaction Manager Facet](#) with the transaction object referenced by the enlistment object referenced by this connection.
- If the connection state is Transaction Aborted:
 - Send a [TXUSER BEGINNER MTAG REQUEST COMPLETED](#) message using the connection.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.1.1.4 Connection Disconnected

When a [CONNTYPE TXUSER BEGINNER \(section 2.2.8.1.1\)](#) connection is disconnected, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is [Active \(section 3.2.1.4.2\)](#):

- Signal the [Begin Rollback \(section 3.2.7.6\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the transaction object referenced by the Enlistment object referenced by this connection.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.4.5.1.2 CONNTYPE_TXUSER_BEGIN2 as Acceptor

For all messages received in this connection type, the transaction manager communicating with an application facet MUST process the message as specified in section [3.1](#). The transaction manager communicating with an application facet MUST also follow the processing rules specified in the following sections.

3.4.5.1.2.1 Receiving a TXUSER_BEGIN2_MTAG_BEGIN Message

When the transaction manager communicating with an application facet receives a [TXUSER_BEGIN2_MTAG_BEGIN \(section 2.2.8.1.2.2\)](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Beginning Transaction.
 - If the transaction manager does not have sufficient memory available to process the message, it MUST:
 - Send a [TXUSER_BEGIN2_MTAG_SINK_ERROR \(section 2.2.8.1.2.5\)](#) message using the connection:
 - The **Error** field MUST be set to TRUN_TXBEGIN_ERROR_NO_MEM.
 - Set the connection state to Ended.
- Otherwise:
 - Create a transaction object using the transaction settings provided in the message:
 - Use the **isoLevel** field as the Isolation Level value of the transaction.
 - Use the **dwTimeout** field as the Timeout value of the transaction.
 - Use the **szDesc** field as the Description value of the transaction.
 - Use the **isoFlags** field as the Isolation Flags value of the transaction.
 - Create a new GUID as specified in [\[RFC4122\]](#) and assign it to the **Identifier** field of the transaction object.
 - Add the connection to the connection list of the transaction.
 - Create a new enlistment object with the following values:
 - The transaction manager communicating with an application facet.
 - The transaction object.
 - The connection.

- Set the **Enlistment** field of the connection to the new enlistment object.
- Signal the [Create Transaction \(section 3.2.7.13\)](#) event on the [Core Transaction Manager Facet](#) with the enlistment object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.1.2.2 Receiving a TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT Message

When the transaction manager communicating with an application facet receives a [TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Active:
 - Set the connection state to Modifying Timeout.
 - Signal the Set Transaction Timeout event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The transaction object referenced by the Enlistment object referenced by this connection.
 - The **dwTxTimeout** field from the message.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.1.2.3 Receiving a TXUSER_BEGIN2_MTAG_COMMIT Message

When the transaction manager communicating with an application facet receives a [TXUSER_BEGIN2_MTAG_COMMIT](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Active:
 - Set the connection state to Committing Transaction.
 - Signal the [Begin Phase Zero \(section 3.6.7.4\)](#) event on the [Core Transaction Manager Facet](#) with the transaction object referenced by the enlistment object referenced by this connection.
- If the connection state is Transaction Aborted:
 - Send a [TXUSER_BEGIN2_MTAG_SINK_ERROR \(section 2.2.8.1.2.5\)](#) message using the connection with the **Error** field set to **TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED**.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.1.2.4 Receiving a TXUSER_BEGIN2_MTAG_ABORT Message

When the transaction manager communicating with an application facet receives a [TXUSER_BEGIN2_MTAG_ABORT \(section 2.2.8.1.2.1\)](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Active:
 - Set the connection state to Aborting transaction.

- Signal the [Begin Rollback \(section 3.4.7.5\)](#) event on the [Core Transaction Manager Facet](#) with the transaction object referenced by the enlistment object referenced by this connection.
- If the connection state is Transaction Aborted:
 - Send a [TXUSER_BEGIN2_MTAG_SINK_ERROR \(section 2.2.8.1.2.5\)](#) message using the connection with the Error field set to **TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED**.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.1.2.5 Connection Disconnected

When a [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) connection is disconnected, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is [Active \(section 3.2.1.4.2\)](#):
 - Signal the [Begin Rollback \(section 3.2.7.6\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the transaction object referenced by the enlistment object referenced by this connection.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.4.5.1.3 CONNTYPE_TXUSER_PROMOTE as Acceptor

For all messages received in this connection type, the transaction manager communicating with an application facet MUST process the message as specified in section [3.1](#). The transaction manager communicating with an application facet MUST also follow the processing rules specified in the following sections.

3.4.5.1.3.1 Receiving a TXUSER_BEGINNER_MTAG_PROMOTE Message

When the transaction manager communicating with an application facet receives a [TXUSER_BEGINNER_MTAG_PROMOTE](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Beginning Transaction.
 - If the transaction manager does not have sufficient memory available to process the message:
 - Send a [TXUSER_BEGIN2_MTAG_SINK_ERROR](#) message using the connection:
 - The Error field MUST be set to **TRUN_TXBEGIN_ERROR_NO_MEM**.
 - Set the connection state to Ended.
 - Otherwise:
 - Create a transaction object using the transaction settings provided in the message:
 - Use the **isoLevel** field as the Isolation Level value of the transaction.
 - Use the **dwTimeout** field as the Timeout value of the transaction.

- Use the **szDesc** field as the Description value of the transaction.
- Use the **isoFlags** field as the Isolation Flags value of the transaction.
- Use the **guidTX** field as the Identifier value of the transaction.
- Add the connection to the connection list of the transaction.
- Create a new enlistment object with the following values:
 - The transaction manager communicating with an application facet.
 - The transaction object.
 - The connection.
- Set the **Enlistment** field of the connection to the new enlistment object.
- Signal the [Create Transaction \(section 3.2.7.13\)](#) event on the [Core Transaction Manager Facet](#) with the enlistment object.
- Otherwise, if the connection state is Active, the message SHOULD be processed as an invalid message as specified in section [3.1.6. <47>](#)
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.1.3.2 Receiving a TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT, TXUSER_BEGIN2_MTAG_COMMIT, or TXUSER_BEGIN2_MTAG_ABORT Message

When the transaction manager communicating with an application facet receives one of these messages, it MUST follow the same message processing rules as the [CONNTYPE_TXUSER_BEGIN2](#) connection type acting as an acceptor, as specified in section [3.4.5.1.2](#).

3.4.5.1.3.3 Connection Disconnected

When a [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#) connection is disconnected, the transaction manager communicating with an application facet MUST perform the same actions as the [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) connection type acting as an acceptor. See section [3.4.5.1.2](#) for more information.

3.4.5.2 Transaction Propagation

3.4.5.2.1 Pull Propagation

3.4.5.2.1.1 CONNTYPE_TXUSER_ASSOCIATE as Acceptor

For all messages received in this connection type, the transaction manager communicating with an application facet MUST process the message as specified in section [3.1](#). The transaction manager MUST also follow the processing rules that are specified in the following sections.

3.4.5.2.1.1.1 Receiving a TXUSER_ASSOCIATE_MTAG_ASSOCIATE Message

When the transaction manager communicating with an application facet receives a [TXUSER_ASSOCIATE_MTAG_ASSOCIATE](#) message, it MUST perform the following actions:

- If the connection state is Idle:

- Override the default schema verification actions for incoming messages as specified in section [3.1.6](#) in the following manner:
 - If the **SourceTmAddr** field from the message violates the constraints specified in section [2.2.4.2](#):
 - Send a [TXUSER_ASSOCIATE_MTAG_CREATE_BAD_TMADDR](#) message using the connection.
 - Perform default invalid message processing, as specified in section [3.1.6](#).
 - Cease processing the message.
 - If the Allow Network Access flag, the Allow Network Transactions flag, or the Allow Inbound Transactions flag of the core transaction manager is set to false:
 - Send a TXUSER_ASSOCIATE_MTAG_CREATE_BAD_TMADDR message using the connection.
 - Set the connection state to Ended.
 - Otherwise, if the transaction manager does not have sufficient memory available to process the message:
 - Send a [TXUSER_ASSOCIATE_MTAG_NO_MEM_LOCAL](#) message.
 - Set the connection state to Ended.
 - Otherwise:
 - Find the transaction object in the transaction table of the transaction manager using the **guidTx** field from the message as the key:
 - If the transaction object is found in the list:
 - Send a [TXUSER_ASSOCIATE_MTAG_ASSOCIATED](#) message to the application.
 - Set the connection state to Ended.
 - Otherwise, if the transaction object is not found in the list, the transaction manager MUST:
 - Set the connection state to Processing Associate Request.
 - Find the list of [CONNTYPE_TXUSER_ASSOCIATE](#) connections in the Associates Table field of the transaction manager communicating with an application, using the **guidTx** field from the message as a key.
 - If the list is found:
 - Add this connection to the list.
 - Otherwise:
 - Create an empty list of CONNTYPE_TXUSER_ASSOCIATE connections and add this connection to it.
 - Add the list to the associates table of the transaction manager communicating with an application under the following key:

- The **guidTx** field from the message.
- Create a new transaction object with the information provided in the message:
 - Use the **guidTx** field as the Identifier value.
 - Use the **isoLevel** field as the Isolation Level value.
 - Use the **isoFlags** field as the Isolation Flags value.
 - Use the **szDesc** field as the Description value.
- Convert the **SourceTmAddr** field from the message to a new Name object, as specified in section [3.1.1.2](#).
- Signal the [Associate Transaction \(section 3.2.7.1\)](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The transaction object
 - The new Name object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.2.1.1.2 Connection Disconnected

When a [CONNTYPE_TXUSER_ASSOCIATE](#) connection is disconnected, the transaction manager communicating with an application facet MUST perform the actions as specified in section [3.1.8.1](#).

3.4.5.2.2 Push Propagation

3.4.5.2.2.1 CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS as Acceptor

For all messages received in this connection type, the transaction manager communicating with an application facet MUST process the message as specified in section [3.1](#). The transaction manager MUST also follow the processing rules specified in the following sections.

3.4.5.2.2.1.1 Receiving a TXUSER_EXTENDEDWHEREABOUTS_MTAG_GET Message

When the transaction manager communicating with an application facet receives a [TXUSER_EXTENDEDWHEREABOUTS_MTAG_GET](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Inquiry.
 - If the transaction manager does not have enough memory to process the TXUSER_EXTENDEDWHEREABOUTS_MTAG_GET message:
 - Send a [TXUSER_EXTENDEDWHEREABOUTS_MTAG_NOMEM](#) message using the connection.
 - Otherwise:
 - Send a [TXUSER_EXTENDEDWHEREABOUTS_MTAG_GOT](#) message using the connection:

- If the **Extended Whereabouts Protocol Count** field of the [Core Transaction Manager Facet](#) is zero:
 - Set the **dwProtocolCount** field to zero.
 - Set the **rgtmprotUsableList** field to empty.
- Otherwise:
 - Set the **dwProtocolCount** field to the **Extended Whereabouts Protocol Count** field of the core transaction manager.
 - Set the contents of the **rgtmprotUsableList** field to the contents of the **Extended Whereabouts** field of the core transaction manager. The size of the **rgtmprotUsableList** field in bytes MUST be determined by the **Extended Whereabouts Size** field of the core transaction manager.
- Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.2.2.1.2 Connection Disconnected

When a [CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS \(section 2.2.8.2.2.1\)](#) connection is disconnected, the transaction manager MUST perform the actions as specified in section [3.1.8.1](#).

3.4.5.2.2.2 CONNTYPE_TXUSER_EXPORT as Acceptor

For all messages received in this connection type, the transaction manager communicating with an application facet MUST process the message as specified in section [3.1](#). The transaction manager MUST also follow the processing rules specified in the following sections.

3.4.5.2.2.2.1 Receiving a TXUSER_EXPORT_MTAG_CREATE Message

When the transaction manager communicating with an application facet receives a [TXUSER_EXPORT_MTAG_CREATE](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Connection Request.
 - Override the default schema verification actions for incoming messages as specified in section [3.1.6](#) in the following manner:
 - If the **SourceTmAddr** field from the message violates the constraints specified in section [2.2.4.2](#), the transaction manager MUST:
 - Send a [TXUSER_EXPORT_MTAG_CREATE_BAD_TMADDR](#) message using the connection.
 - Perform default invalid message processing, as specified in section [3.1.6](#).
 - Cease processing the message.
 - If the Allow Network Access flag, the Allow Network Transactions flag, or the Allow Outbound Transactions flag of the [Core Transaction Manager Facet](#) is set to false:

- Send a `TXUSER_EXPORT_MTAG_CREATE_BAD_TMADDR` message using the connection.
- Set the connection state to Ended.
- Otherwise:
 - Convert the **SourceTmAddr** field from the message to a Name object, as specified in section [3.1.1](#).
 - Store the Name object in the **Connection-Specific Data** field of the connection object.
 - Send a `TXUSER_EXPORT_MTAG_CREATED` message using the connection.
 - Set the connection state to Connection Active.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.2.2.2.2 Receiving a TXUSER_EXPORT_MTAG_CREATE2 Message

When the transaction manager receives a `TXUSER_EXPORT_MTAG_CREATE2` message, the transaction manager MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Connection Request.
 - Override the default schema verification actions for incoming messages as specified in section [3.1.6](#) in the following manner:
 - If the SourceTmAddr field in the message does not comply with the constraints specified in section [2.2.4.2](#):
 - Send a `TXUSER_EXPORT_MTAG_CREATE_BAD_TMADDR` message using the connection.
 - Perform default invalid message processing, as specified in section [3.1.6](#).
 - Cease processing the message.
 - If the Allow Network Access flag, the Allow Network Transactions flag, or the Allow Outbound Transactions flag of the [Core Transaction Manager Facet](#) is set to false:
 - Send a `TXUSER_EXPORT_MTAG_CREATE_NET_TX_DISABLED` message using the connection.
 - Set the connection state to Ended.
 - Otherwise:
 - Send a `TXUSER_EXPORT_MTAG_CREATED` message using the connection.
 - Set the connection state to Connection Active.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.2.2.2.3 Receiving a TXUSER_EXPORT_MTAG_EXPORT Message

When the transaction manager receives a `TXUSER_EXPORT_MTAG_EXPORT` message, the transaction manager MUST perform the following actions:

- If the connection state is Connection Active:
 - Find the transaction object in the transaction table of the transaction manager by using the **guidTX** field from the message as the key.
 - If the transaction object is not found:
 - Send a [TXUSER_EXPORT_MTAG_EXPORT_TX_NOT_FOUND](#) message using the connection.
 - Set the connection state to Ended.
 - Otherwise:
 - Set the connection state to Processing Push Operation Request.
 - Add the connection to the connection list of the transaction.
 - Signal the [Export Transaction \(section 3.2.7.21\)](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The Name object stored in the **Connection-Specific Data** field of the connection.
 - The transaction object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.2.2.2.4 Connection Disconnected

When a [CONNTYPE_TXUSER_EXPORT \(section 2.2.8.2.2.2\)](#) connection is disconnected, the transaction manager communicating with an application facet MUST perform the actions as specified in section [3.1.8.1](#).

3.4.5.2.2.3 CONNTYPE_TXUSER_IMPORT as Acceptor

For all messages received in this connection type, the transaction manager communicating with an application facet MUST process the messages as specified in section [3.1](#). The transaction manager MUST also follow the processing rules specified in the following sections.

3.4.5.2.2.3.1 Receiving a_TXUSER_IMPORT_MTAG_IMPORT Message

When the transaction manager receives a [TXUSER_IMPORT_MTAG_IMPORT](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Import Request.
 - Find the transaction object in the transaction table of the transaction manager by using the **guidTx** field from the message as the key.
 - If the transaction object is not found or if the transaction state is not Active, Phase Zero or Phase Zero Complete:
 - Send a [TXUSER_IMPORT_MTAG_IMPORT_TX_NOT_FOUND](#) message using the connection.
 - Set the connection state to Ended.
- Otherwise:

- Set the connection state to Active.
- Add the connection to the connection list of the transaction.
- Create a new Enlistment object using the following fields:
 - The transaction manager communicating with an application facet.
 - The transaction object.
 - The connection object.
- Assign the new Enlistment object to the **Enlistment** field of the connection.
- Signal the [Create Voter Enlistment](#) event on the [Core Transaction Manager Facet](#) with the new Enlistment object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.2.2.3.2 Receiving a_TXUSER_IMPORT_MTAG_ABORT Message

When the transaction manager receives a [TXUSER_IMPORT_MTAG_ABORT](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Voted:
 - Send a [TXUSER_IMPORT_MTAG_ABORT_TOO_LATE](#) message using the connection.
- If the connection state is Active:
 - Set the connection state to Processing Abort Request.
 - Signal the [Enlistment Unilaterally Aborted \(section 3.2.7.19\)](#) message of the [Core Transaction Manager Facet](#) with the **Enlistment** field of the connection.
 - Send a TXUSER_IMPORT2_MTAG_SINK_ERROR message.
 - The **Error** field MUST be set to TRUN_TXIMPORT_ERROR_NOTIFY_ABORTED.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.2.2.3.3 Connection Disconnected

When a [CONNTYPE_TXUSER_ASSOCIATE \(section 2.2.8.2.1.1\)](#) connection is disconnected, the transaction manager MUST perform the actions as specified in section [3.1.8.1](#).

3.4.5.2.2.4 CONNTYPE_TXUSER_IMPORT2 as Acceptor

For all messages received in this connection type, the transaction manager communicating with an application facet MUST process the messages as specified in section [3.1](#). The transaction manager MUST also follow the processing rules specified in the following sections.

3.4.5.2.2.4.1 Receiving a TXUSER_IMPORT2_MTAG_IMPORT Message

When the transaction manager receives a [TXUSER_IMPORT2_MTAG_IMPORT](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Import Request.
 - Find the transaction object in the transaction table of the transaction manager by using the **guidTx** field from the message as the key.
 - If the transaction object is not found or if the transaction state is not Active, Phase Zero or Phase Zero Complete:
 - Send a [TXUSER_IMPORT2_MTAG_SINK_ERROR](#) message using the connection:
 - The **Error** field MUST be set to TRUN_TXIMPORT_ERROR_IMPORT_TX_NOT_FOUND.
 - Set the connection state to Ended.
 - Otherwise:
 - Add the connection to the connection list of the transaction.
 - Create a new Enlistment object using the following fields:
 - The transaction manager communicating with an application facet.
 - The transaction object.
 - The connection object.
 - Assign the new Enlistment object to the **Enlistment** field of the connection.
 - Signal the [Create Voter Enlistment](#) event on the [Core Transaction Manager Facet](#) with the new Enlistment object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.2.2.4.2 Receiving a TXUSER_IMPORT2_MTAG_IMPORT_WITH_SET Message

When the transaction manager receives a [TXUSER_IMPORT2_MTAG_IMPORT_WITH_SET](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Import Request.
 - Find the transaction object in the transaction table of the transaction manager by using the **guidTx** field from the message as the key.
 - If the transaction object is not found or if the transaction state is not Active, Phase Zero, or Phase Zero Complete:
 - Send a [TXUSER_IMPORT2_MTAG_SINK_ERROR](#) message using the connection:
 - The **Error** field MUST be set to TRUN_TXIMPORT_ERROR_IMPORT_TX_NOT_FOUND.
 - Set the connection state to Ended.
 - Otherwise:
 - Add the connection to the connection list of the transaction.

- Signal the [Set Transaction Attributes \(section 3.2.7.31\)](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The transaction object
 - The **isoLevel** field from the message.
 - The **isoFlags** field from the message.
 - The **szDesc** field from the message.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.2.2.4.3 Receiving a TXUSER_IMPORT2_MTAG_ABORT Message

When the transaction manager receives a [TXUSER_IMPORT2_MTAG_ABORT \(section 2.2.8.2.2.4.1\)](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Voted:
 - Send a [TXUSER_IMPORT2_MTAG_SINK_ERROR \(section 2.2.8.2.2.4.4\)](#) message using the connection:
 - The **Error** field MUST be set to TRUN_TXIMPORT_ERROR_IMPORT_TX_NOT_FOUND.
- If the connection state is Active:
 - Set the connection state to Processing Abort Request.
 - Signal the [Enlistment Unilaterally Aborted \(section 3.2.7.19\)](#) event of the [Core Transaction Manager Facet](#) with the **Enlistment** field of the connection.
 - Send a TXUSER_IMPORT_MTAG_REQUEST_COMPLETED message.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.2.2.4.4 Connection Disconnected

When a [CONNTYPE_TXUSER_ASSOCIATE \(section 2.2.8.2.1.1\)](#) connection is disconnected, the transaction manager MUST perform the actions as specified in section [3.1.8.1](#).

3.4.5.3 Transaction Administration

3.4.5.3.1 CONNTYPE_TXUSER_GETTXDETAILS as Acceptor

For all messages received in this connection type, the transaction manager MUST process the message as specified in section [3.1](#). The transaction manager MUST also follow the processing rules that are specified in the following sections.

3.4.5.3.1.1 Receiving a TXUSER_GETTXDETAILS_MTAG_GET Message

When the transaction manager receives a [TXUSER_GETTXDETAILS_MTAG_GET](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Idle:

- Set the connection state to Processing Inquiry.
- Find the transaction object in the transaction table of the transaction manager by using the **guidTx** field from the message as the key:
 - If the transaction object is not found in the list, the transaction manager MUST:
 - Send a [TXUSER_GETTXDETAILS_MTAG_TX_NOT_FOUND](#) message using the connection.
 - Otherwise:
 - Send a [TXUSER_GETTXDETAILS_MTAG_GOTIT](#) message using the connection with the message fields set as follows:
 - The **vszSuperiorName** field MUST be set to the transaction object's Superior Enlistment object's Name property.
 - The **vszSuperiorID** field MUST be set to the transaction object's Superior Enlistment object's Identifier property.
 - The **rgSubordinates** field Subordinates MUST be set to an array of OLETX_VARLEN_STRING structures. Each Subordinate entry is represented by two adjacent structures, whose values are set as follows:
 - For each Enlistment object in the Phase One Enlistment and Phase Two Enlistment lists of the transaction:
 - The first subordinate structure MUST be set to the Name property of the Enlistment object.
 - The second subordinate structure MUST be set to the Identifier property of the Enlistment object.
 - The **ISubordinateCount** field MUST be set to the number of Enlistment objects whose values were added to the rgSubordinates array.
- Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1](#).

3.4.5.3.1.2 Connection Disconnected

When a [CONNTYPE_TXUSER_GETTXDETAILS \(section 2.2.8.3.1\)](#) connection is disconnected, the transaction manager MUST perform the actions as specified in section [3.1.8.1](#).

3.4.5.3.2 CONNTYPE_TXUSER_RESOLVE as Acceptor

For all messages received in this connection type, the transaction manager communicating with an application facet MUST process the message as specified in section [3.1](#). The transaction manager MUST also follow the processing rules specified in the following sections.

3.4.5.3.2.1 Receiving a TXUSER_RESOLVE_MTAG_CHILD_ABORT Message

When the transaction manager communicating with an application facet receives a [TXUSER_RESOLVE_MTAG_CHILD_ABORT](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Abort Request.
 - Find the transaction object in the transaction table of the transaction manager by using the **guidTx** field provided in the message as a key.
 - If the transaction object is not found:
 - Send a [TXUSER_RESOLVE_MTAG_TX_NOT_FOUND](#) message by using the connection.
 - Set the connection state to Ended.
 - Otherwise:
 - Add the connection to the transactionconnection list.
 - Signal the [Resolve Transaction \(section 3.2.7.30\)](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The transaction object.
 - The Aborted outcome.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.3.2.2 Receiving a TXUSER_RESOLVE_MTAG_CHILD_COMMIT Message

When the transaction manager communicating with an application facet receives a [TXUSER_RESOLVE_MTAG_CHILD_COMMIT](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Commit request.
 - Find the transaction object in the transaction table of the transaction manager by using the **guidTx** field provided in the message as a key.
 - If the transaction object is not found:
 - Send a [TXUSER_RESOLVE_MTAG_TX_NOT_FOUND](#) message by using the connection.
 - Set the connection state to Ended.
 - Otherwise:
 - Add the connection to the transaction connection list.
 - Signal the Resolve Transaction event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The transaction object
 - The Committed outcome.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.3.2.3 Receiving a TXUSER_RESOLVE_MTAG_FORGET_COMMITTED Message

When the transaction manager communicating with an application facet receives a [TXUSER_RESOLVE_MTAG_FORGET_COMMITTED](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Forget Request.
 - Find the transaction object in the transaction table of the transaction manager by using the **guidTx** field provided in the message as a key.
 - If the transaction object is not found:
 - Send a [TXUSER_RESOLVE_MTAG_TX_NOT_FOUND](#) using the connection.
 - Set the connection state to Ended.
 - Otherwise:
 - Add the connection to the transaction connection list.
 - Signal the [Resolve Transaction \(section 3.2.7.30\)](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The transaction object.
 - The Forgotten outcome.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.3.2.4 Connection Disconnected

When a [CONNTYPE_TXUSER_RESOLVE \(section 2.2.8.3.2\)](#) connection is disconnected, the transaction manager MUST perform the actions as specified in section [3.1.8.1](#).

3.4.5.3.3 CONNTYPE_TXUSER_SETTXTIMEOUT as Acceptor

For all messages received in this connection type, the transaction manager MUST process the message as specified in section [3.1](#). The transaction manager MUST also follow the processing rules specified in the following sections.

3.4.5.3.3.1 Receiving a TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT Message

When the transaction manager receives a [TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT \(section 2.2.8.1.2.7\)](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Active:
 - Set the connection state to Resetting Timeout.
 - Find the transaction object in the transaction table of the transaction manager by using the **guidTx** field from the message as the key.
 - If the transaction object is not found in the list:

- Send the application a [TXUSER_SETTXTIMEOUT MTAG TX NOT FOUND \(section 2.2.8.3.3.1\)](#) message.
- Set the connection state to Ended.
- Otherwise:
 - Signal the [Set Transaction Timeout \(section 3.2.7.32\)](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The transaction object.
 - The **dwTxTimeout** field from the message.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.4.5.3.3.2 Connection Disconnected

When a [CONNTYPE_TXUSER_SETTXTIMEOUT \(section 2.2.8.3.3\)](#) connection is disconnected, the transaction manager MUST perform the actions as specified in section [3.1.8.1](#).

3.4.5.3.4 CONNTYPE_TXUSER_SETTXTIMEOUT2 as Acceptor

For all messages received in this connection type, the transaction manager MUST process the message as specified in section [3.1](#). The application MUST also follow the processing rules specified in the following sections.

3.4.5.3.4.1 Receiving a TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT Message

When the transaction manager receives a [TXUSER_SETTXTIMEOUT_MTAG_SETTXTIMEOUT](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Request.
 - Send a [TXUSER_SETTXTIMEOUT_MTAG_TX_NOT_FOUND](#) message.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as specified in section [3.1.6](#).

3.4.5.3.4.2 Connection Disconnected

When a [CONNTYPE_TXUSER_SETTXTIMEOUT \(section 2.2.8.3.3\)](#) connection is disconnected, the transaction manager MUST perform the actions as specified in section [3.1.8.1](#).

3.4.5.3.5 CONNTYPE_TXUSER_TRACE as Acceptor

For all messages received in this connection type, the transaction manager communicating with an application facet MUST process the message as specified in section [3.1](#). The transaction manager communicating with an application facet MUST also follow the processing rules specified in the following sections.

3.4.5.3.5.1 Receiving a TXUSER_TRACE_MTAG_DUMP_TRANSACTION Message

When the transaction manager communicating with an application facet receives a [TXUSER_TRACE_MTAG_DUMP_TRANSACTION](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Trace Request.
 - Find the transaction object in the transaction table of the transaction manager by using the **guidTx** field from the message as the key.
 - If the transaction object is not found in the list, the transaction manager MUST:
 - Send a [TXUSER_TRACE_MTAG_TX_NOT_FOUND](#) message using the connection.
 - Otherwise:
 - Attempt to generate trace records for the transaction in the trace file of the transaction manager in an implementation-specific manner.
 - If the operation fails:
 - Send a [TXUSER_TRACE_MTAG_REQUEST_FAILED](#) message using the connection.
 - Otherwise:
 - Send a [TXUSER_TRACE_MTAG_REQUEST_COMPLETE](#) message using the connection.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.3.5.2 Connection Disconnected

When a [CONNTYPE_TXUSER_TRACE \(section 2.2.8.3.5\)](#) connection is disconnected, the transaction manager communicating with an application facet MUST perform the actions as specified in section [3.1.8.1](#).

3.4.5.4 Transaction Manager Administration

3.4.5.4.1 CONNTYPE_TXUSER_GETSECURITYFLAGS as Acceptor

For all messages received in this connection type, the transaction manager communicating with an application facet MUST process the message as specified in section [3.1](#). The transaction manager communicating with an application facet MUST also follow the processing rules specified in the following sections.

3.4.5.4.1.1 Receiving a TXUSER_GETSECURITYFLAGS_MTAG_GETSECURITYFLAGS Message

When the transaction manager communicating with an application facet receives a [TXUSER_GETSECURITYFLAGS_MTAG_GETSECURITYFLAGS](#) message, the transaction manager communicating with an application facet MUST perform the following actions:

- If the connection state is Idle:

- Set the connection state to Processing Request.
- Send a [TXUSER GETSECURITYFLAGS MTAG FETCHED](#) message using the connection:
 - If the Allow Network Access flag of the [Core Transaction Manager Facet](#) is set to false:
 - Set the **grfNetworkDtcAccess** field to zero.
 - Otherwise, set the **grfNetworkDtcAccess** field as follows:
 - Set all bits to zero by default.
 - Set the DTCADVCONFIG_NETWORKDTCACCESS_ENABLE bit to 1.
 - If the Allow Remote Administration flag of the Core Transaction Manager Facet is set to true:
 - Set the DTCADVCONFIG_NETWORKDTCACCESS_ADMIN bit to 1.
 - If the Allow Network Transactions flag of the Core Transaction Manager Facet is set to true:
 - Set the DTCADVCONFIG_NETWORKDTCACCESS_TX bit to 1.
 - If the Allow Remote Clients flag of the Core Transaction Manager Facet is set to true:
 - Set the DTCADVCONFIG_NETWORKDTCACCESS_CLIENTS bit to 1.
 - If the Allow TIP flag of the Core Transaction Manager Facet is set to true:
 - Set the DTCADVCONFIG_NETWORKDTCACCESS_TIP bit to 1.
 - If the Allow Outbound Transactions flag of the Core Transaction Manager Facet is set to true:
 - Set the DTCADVCONFIG_OUTBOUNDNETWORK_TX bit to 1.
 - If the Allow Inbound Transactions flag of the Core Transaction Manager Facet is set to true:
 - Set the DTCADVCONFIG_INBOUNDNETWORK_TX bit to 1.
 - If the **Security Level** field of the Core Transaction Manager Facet is set to no security:
 - Set the DTCADVCONFIG_SECURITYLEVEL_NOSECURITY bit to 1.
 - Otherwise, if the **Security Level** field of the Core Transaction Manager Facet is set to incoming authentication:
 - Set the DTCADVCONFIG_SECURITYLEVEL_AUTHENTICATEDONLY bit to 1.
 - Otherwise, if the **Security Level** field of the Core Transaction Manager Facet is set to mutual authentication:
 - Set the DTCADVCONFIG_SECURITYLEVEL_MUTUALAUTH bit to 1.
 - If the Allow XA flag of the Core Transaction Manager Facet is set to true, set the **grfXaTransaction** field to 1; otherwise, set the flag to zero.

- Set the **grfReserved3** field to zero.
- Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.4.5.4.1.2 Connection Disconnected

When a [CONNTYPE_TXUSER_TRACE \(section 2.2.8.3.5\)](#) connection is disconnected, the transaction manager communicating with an application facet MUST perform the actions as specified in section [3.1.8.1](#).

3.4.6 Timer Events

No timer events apply here.

3.4.7 Other Local Events

A transaction manager communicating with an application facet MUST be prepared to process the local events that are defined in the following sections.

The transaction manager communicating with an application facet MUST be prepared to process local events pertaining to [Phase Zero](#) functionality only on versions where the connection type CONNTYPE_TXUSER_PHASE0 is supported. Section [2.2.1.1.3](#) defines protocol version support for this connection type. The following local events are affected:

- [Register Phase Zero \(section 3.4.7.15\)](#)
- [Phase Zero Complete \(section 3.4.7.14\)](#)

3.4.7.1 Associate Transaction Failure

The Associate Transaction Failure event MUST be signaled with the following arguments:

- A transaction object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:
 - Comm Failed
 - Log Full Remote
 - Log Full Local
 - No Mem Remote
 - Too Late
 - Too Many Remote
 - Too Many Local
 - Tx Not Found

If the Associate Transaction Failure event is signaled, the transaction manager MUST perform the following actions:

- Find an instance of a [CONNTYPE TXUSER ASSOCIATE \(section 2.2.8.2.1.1\)](#) connection list in the associates table of the transaction manager communicating with an application facet by using the identifier of the transaction object as the key.
- For each connection in the list:
 - Remove the connection from the list.
 - If the connection state is Processing Associate Request:
 - Send the matching message for the following reason codes:
 - Comm Failed: [TXUSER ASSOCIATE MTAG COMM FAILED \(section 2.2.8.2.1.1.3\)](#)
 - Log Full Remote: [TXUSER ASSOCIATE MTAG LOG FULL REMOTE \(section 2.2.8.2.1.1.6\)](#)
 - Log Full Local: [TXUSER ASSOCIATE MTAG LOG FULL LOCAL \(section 2.2.8.2.1.1.5\)](#)
 - No Mem Remote: [TXUSER ASSOCIATE MTAG NO MEM REMOTE \(section 2.2.8.2.1.1.8\)](#)
 - Too Late: [TXUSER ASSOCIATE MTAG TOO LATE \(section 2.2.8.2.1.1.9\)](#)
 - Too Many Remote: [TXUSER ASSOCIATE MTAG TOO MANY REMOTE \(section 2.2.8.2.1.1.11\)](#)
 - Too Many Local: [TXUSER ASSOCIATE MTAG TOO MANY LOCAL \(section 2.2.8.2.1.1.10\)](#)
 - Tx Not Found: [TXUSER ASSOCIATE MTAG TX NOT FOUND \(section 2.2.8.2.1.1.12\)](#)
 - Set the connection state to Ended.
- Remove the list from the associates table of the transaction manager communicating with an application facet.

3.4.7.2 Associate Transaction Success

The Associate Transaction Success event MUST be signaled with the following arguments:

- A transaction object.

If the Associate Transaction Success event is signaled, the transaction manager MUST perform the following actions:

- Find the list of [CONNTYPE TXUSER ASSOCIATE \(section 2.2.8.2.1.1\)](#) connections in the associates table of the transaction manager communicating with an application facet by using the identifier of the transaction object as the key.
- For each connection in the list:
 - Remove the connection from the list.
 - If the connection state is Processing Associate Request:
 - Send a [TXUSER ASSOCIATE MTAG ASSOCIATED \(section 2.2.8.2.1.1.2\)](#) message using the connection.

- Set the connection state to Active.
- Remove the list from the associates table of the transaction manager communicating with an application facet.

3.4.7.3 Begin Commit

The Begin Commit event MUST be signaled with the following arguments:

- An Enlistment object.

If the Begin Commit event is signaled, the transaction manager MUST perform the following actions:

- If the enlistment connection is of type [CONNTYPE_TXUSER_IMPORT \(section 2.2.8.2.2.3\)](#):
 - Set the connection state to [Ended \(section 3.2.1.4.14\)](#).
- Otherwise, if the enlistment connection is of type [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.2.4\)](#) or [CONNTYPE_TXUSER_ASSOCIATE \(section 2.2.8.2.1.1\)](#):
 - Send a [TXUSER_IMPORT2_MTAG_SINK_ERROR \(section 2.2.8.2.2.4.4\)](#) message:
 - The **Error** field MUST be set to TRUN_TXIMPORT_ERROR_NOTIFY_COMMITTED.
 - Set the connection state to Ended.

3.4.7.4 Begin In Doubt

The [Begin In Doubt](#) event MUST be signaled with the following arguments:

- An enlistment object.

If the Begin In Doubt event is signaled, the transaction manager MUST perform the following actions:

- If the enlistment connection is of type [CONNTYPE_TXUSER_IMPORT \(section 2.2.8.2.2.3\)](#):
 - Set the connection state to Ended.
- Otherwise, if the enlistment connection is of type [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.2.4\)](#) or [CONNTYPE_TXUSER_ASSOCIATE \(section 2.2.8.2.1.1\)](#):
 - Send a [TXUSER_IMPORT2_MTAG_SINK_ERROR \(section 2.2.8.2.2.4.4\)](#) message:
 - The **Error** field MUST be set to TRUN_TXIMPORT_ERROR_NOTIFY_INDOUBT.
 - Set the connection state to Ended.

3.4.7.5 Begin Rollback

The Begin Rollback event MUST be signaled with the following arguments:

- An enlistment object.

If the Begin Rollback event is signaled, the transaction manager MUST perform the following actions:

- If the enlistment's connection is of type [CONNTYPE_TXUSER_IMPORT \(section 2.2.8.2.2.3\)](#):

- Set the connection state to Ended.
- Otherwise, if the enlistment's connection is of type [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.2.4\)](#) or [CONNTYPE_TXUSER_ASSOCIATE \(section 2.2.8.2.1.1\)](#):
 - If the connection state is Active or Too Late to Abort:
 - Send a [TXUSER_IMPORT2_MTAG_SINK_ERROR \(section 2.2.8.2.2.4.4\)](#) message:
 - The **Error** field MUST be set to TRUN_TXIMPORT_ERROR_NOTIFY_ABORTED.
 - Set the connection state to Ended.

3.4.7.6 Begin Voting

The Begin Voting Success event MUST be signaled with the following arguments:

- An Enlistment object.

If the Begin Voting event is signaled, the transaction manager MUST perform the following actions:

- If the enlistment's connection is of type [CONNTYPE_TXUSER_IMPORT](#), [CONNTYPE_TXUSER_IMPORT2](#) or [CONNTYPE_TXUSER_ASSOCIATE](#):
 - Signal the Enlistment Vote Complete event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The provided Enlistment object.
 - The Prepared vote outcome.
 - If the enlistment's connection type is CONNTYPE_TXUSER_IMPORT or CONNTYPE_TXUSER_IMPORT2:
 - Set the connection state to Too Late to Abort.

3.4.7.7 Create Transaction Failure

The Create Transaction Failure event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:
 - Log Full
 - No Mem
 - Duplicate

If the Create Transaction Failure event is signaled, the transaction manager MUST perform the following actions:

- If the transaction's connection list contains a connection of type [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#):
 - Send the matching message for the following reason codes using the provided enlistment's connection:

- Log Full: [TXUSER BEGINNER MTAG BEGIN LOG FULL \(section 2.2.8.1.1.3\)](#).
- No Mem: [TXUSER BEGINNER MTAG BEGIN NO MEM \(section 2.2.8.1.1.4\)](#).
- Duplicate: TXUSER_BEGINNER_MTAG_BEGIN_LOG_FULL (section 2.2.8.1.1.3).
- Set the connection state to Ended.
- Otherwise, if the transaction's connection list contains a connection of type [CONNTYPE TXUSER BEGIN2 \(section 2.2.8.1.2\)](#) or [CONNTYPE TXUSER PROMOTE \(section 2.2.8.1.3\)](#):
 - Send a [TXUSER BEGIN2 MTAG SINK ERROR \(section 2.2.8.1.2.5\)](#) message using the provided enlistment's connection:
 - The Error field MUST be set to the value matching the following reason codes:
 - Log Full: TRUN_TXBEGIN_ERROR_BEGIN_LOG_FULL.
 - No Mem: TRUN_TXBEGIN_ERROR_NO_MEM.
 - Duplicate: TRUN_TXBEGIN_ERROR_DUPLICATE_GUID<48>.
 - Set the connection state to Ended.

3.4.7.8 Create Transaction Success

The Create Transaction Success event MUST be signaled with the following arguments:

- A Transaction object.

If the Create Transaction Success event is signaled, the transaction manager MUST perform the following actions:

- If the transaction's connection list contains a connection of type [CONNTYPE TXUSER BEGINNER \(section 2.2.8.1.1\)](#)
 - Send a [TXUSER BEGIN2 MTAG SINK BEGUN \(section 2.2.8.1.2.4\)](#) message.
 - The **guidTx** field MUST be set to the transaction's identifier.
 - Set the connection state to Active.
- Otherwise, if the transaction's connection list contains a connection of type [CONNTYPE TXUSER BEGIN2 \(section 2.2.8.1.2\)](#) or [CONNTYPE TXUSER PROMOTE \(section 2.2.8.1.3\)](#):
 - Send a TXUSER_BEGIN2_MTAG_SINK_BEGUN (section 2.2.8.1.2.4) message:
 - The **guidTx** field MUST be set to the transaction's identifier.
 - Set the connection state to Active.

3.4.7.9 Create Voter Enlistment Failure

The Create Voter Enlistment Failure event MUST be signaled with the following arguments:

- An Enlistment object.

- A value indicating the failure reason. The reason MUST be set to the following value:
 - Too Late

If the Create Voter Enlistment Failure event is signaled, the Transaction Manager MUST perform the following actions:

- If the provided enlistment's connection is of type [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.2.4\)](#):
 - Send a [TXUSER_IMPORT2_MTAG_SINK_IMPORTED \(section 2.2.8.2.2.4.5\)](#) message using the provided enlistment's connection:
 - The **Error** field MUST be set to:
 - TRUN_TXIMPORT_ERROR_IMPORT_TX_NOT_FOUND.
 - Set the connection state to Ended.
- Otherwise, if the provided enlistment's connection is of type [CONNTYPE_TXUSER_IMPORT \(section 2.2.8.2.2.3\)](#):
 - Send a TXUSER_IMPORT_MTAG_IMPORT_TX_NOT_FOUND message using the provided enlistment's connection.
 - Set the connection state to Ended.

3.4.7.10 Create Voter Enlistment Success

The Create Voter Enlistment Success event MUST be signaled with the following arguments:

- An Enlistment object

If the Create Voter Enlistment Success event is signaled, the Transaction Manager MUST perform the following actions:

- If the provided enlistment's connection is of type [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.2.4\)](#):
 - Send the [TXUSER_IMPORT2_MTAG_SINK_IMPORTED \(section 2.2.8.2.2.4.5\)](#) using the provided enlistment's connection.
- Otherwise, if the provided enlistment's connection is of type [CONNTYPE_TXUSER_IMPORT \(section 2.2.8.2.2.3\)](#):
 - Send the [TXUSER_IMPORT_MTAG_IMPORTED \(section 2.2.8.2.2.3.5\)](#) using the provided enlistment's connection:
- Set the connection state to Ended.

3.4.7.11 Export Transaction Failure

The Export Transaction Failure event MUST be signaled with the following arguments:

- A Transaction object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:

- Log Full
- No Mem
- Too Late
- Too Many
- Tx Not Found

If the Export Transaction Failure event is signaled, the transaction manager MUST perform the following actions:

- Find an instance of a [CONNTYPE_TXUSER_EXPORT \(section 2.2.8.2.2.2\)](#) connection in the provided transaction's connection list.
- If no such connection exists, the event MUST be ignored.
- Otherwise:
 - Send the matching message for the following reason codes:
 - Log Full: [TXUSER_EXPORT_MTAG_EXPORT_LOG_FULL \(section 2.2.8.2.2.2.7\)](#).
 - No Mem: [TXUSER_EXPORT_MTAG_EXPORT_NO_MEM \(section 2.2.8.2.2.2.8\)](#).
 - Too Late: [TXUSER_EXPORT_MTAG_EXPORT_TOO_LATE \(section 2.2.8.2.2.2.9\)](#).
 - Too Many: [TXUSER_EXPORT_MTAG_EXPORT_TOO_MANY \(section 2.2.8.2.2.2.10\)](#).
 - Tx Not Found: [TXUSER_EXPORT_MTAG_EXPORT_TX_NOT_FOUND \(section 2.2.8.2.2.2.11\)](#).
- Set the connection state to Ended.

3.4.7.12 Export Transaction Success

The Export Transaction Success event MUST be signaled with the following arguments:

- A Transaction object.

If the Export Transaction Success event is signaled, the transaction manager MUST perform the following actions:

- Find an instance of a [CONNTYPE_TXUSER_EXPORT \(section 2.2.8.2.2.2\)](#) connection in the provided transaction's connection list.
- If no such connection exists, the event MUST be ignored.
- Otherwise:
 - Send a [TXUSER_EXPORT_MTAG_EXPORTED \(section 2.2.8.2.2.2.12\)](#) message using the connection.
- Set the connection state to Active.

3.4.7.13 Phase One Complete

The Phase One Complete event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the outcome of [Phase One](#). The value MUST be set to one of the following values:
 - Read Only
 - Committed
 - Aborted
 - In Doubt

If the Phase One Complete event is signaled, the Transaction Manager Communicating with an ApplicationFacet MUST perform the following actions:

- If the provided outcome is Read Only or Committed:
 - If the transaction's connection list contains a connection of type [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#):
 - Send a [TXUSER_BEGINNER_MTAG_REQUEST_COMPLETED \(section 2.2.8.1.1.9\)](#) message.
 - Set the connection state to Ended.
 - Otherwise, if the transaction's connection list contains a connection of type [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) or [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#):
 - Send a [TXUSER_BEGIN2_MTAG_SINK_ERROR \(section 2.2.8.1.2.5\)](#) message:
 - The Error field MUST be set to TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED.
 - Set the connection state to Ended.
- Otherwise, if the provided outcome is Aborted:
 - If the transaction's connection list contains a connection of type [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#):
 - If the connection state is Active:
 - Set the connection state to Transaction Aborted.
 - Otherwise, if the connection state is Aborting Transaction or Committing Transaction:
 - Send a [TXUSER_BEGINNER_MTAG_REQUEST_COMPLETED \(section 2.2.8.1.1.9\)](#) message.
 - Set the connection state to Ended.
 - Otherwise, ignore the event.
 - Otherwise, if the transaction's connection list contains a connection of type [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) or [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#):
 - Send a [TXUSER_BEGIN2_MTAG_SINK_ERROR \(section 2.2.8.1.2.5\)](#) message:

- The Error field MUST be set to TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED.
- Set the connection state to Ended.
- Otherwise, if the provided outcome is [In Doubt](#):
 - If the transaction's connection list contains a CONNTYPE_TXUSER_BEGINNER (section 2.2.8.1.1) connection:
 - Send a [TXUSER_BEGINNER_MTAG_COMMIT_INDOUBT \(section 2.2.8.1.1.7\)](#) message.
 - Set the connection state to Ended.
 - Otherwise, if the transaction's connection list contains a CONNTYPE_TXUSER_BEGIN2 (section 2.2.8.1.2) connection:
 - Send a TXUSER_BEGIN2_MTAG_SINK_ERROR (section 2.2.8.1.2.5) message:
 - The Error field MUST be set to TRUN_TXBEGIN_ERROR_NOTIFY_INDOUBT.
 - Set the connection state to Ended.

3.4.7.14 Phase Zero Complete

The Phase Zero Complete event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the outcome of [Phase Zero](#). The value MUST be set to one of the following values:
 - Success
 - Failure

If the Phase Zero Complete event is signaled, the transaction manager MUST perform the following actions:

- If the provided outcome is Success:
 - Signal the Begin Phase One event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The Transaction referenced by the provided Enlistment object.
 - The Single Phase Commit flag set to true.

Otherwise:

- Signal the Phase One Complete event on the Transaction Manager Communicating with an ApplicationFacet with the following arguments:
 - The provided Enlistment object.
 - The Aborted outcome.

3.4.7.15 Register Phase Zero

The Register Phase Zero event MUST be signaled with the following arguments:

- An Enlistment object.

If the Register Phase Zero event is signaled, the transaction manager MUST perform the following actions:

- Signal the Register Phase Zero Success event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The provided Enlistment object.

3.4.7.16 Resolve Transaction Complete

The Resolve Transaction Complete event MUST be signaled with the following arguments:

- A Transaction object.
- A value indicating the result of the resolve transaction operation. The value MUST be set to one of the following values:
 - Committed
 - Aborted
 - Forgotten
 - Not Prepared
 - Not Committed

If the Resolve Transaction Complete event is signaled, the transaction manager MUST perform the following actions:

- Find a [CONNTYPE_TXUSER_RESOLVE \(section 2.2.8.3.2\)](#) connection in the transaction's connection list.
- If the connection is not found, ignore the signal.
- Otherwise
 - If the resolve outcome is Committed, Aborted, or Forgotten,
 - Send a [TXUSER_RESOLVE_MTAG_REQUEST_COMPLETE \(section 2.2.8.3.2.7\)](#) message using the connection.
 - Set the connection state to Ended.
 - Otherwise, if the resolve outcome is Not Prepared:
 - Send a [TXUSER_RESOLVE_MTAG_CHILD_NOT_PREPARED \(section 2.2.8.3.2.4\)](#) message using the connection.
 - Set the connection state to Ended.
 - Otherwise, if the resolve outcome is Not Committed:

- Send a [TXUSER_RESOLVE_MTAG_FORGET_TX_NOT_COMMITTED \(section 2.2.8.3.2.6\)](#) message using the connection.
- Set the connection state to Ended.

3.4.7.17 Rollback Complete

The Rollback Complete event MUST be signaled with the following arguments:

- An Enlistment object.

If the Rollback Complete event is signaled, the transaction manager MUST perform the following actions:

- If the Connection referenced by the Enlistment is of type [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#):
 - If the Connection state is Aborting Transaction:
 - Send a [TXUSER_BEGINNER_MTAG_REQUEST_COMPLETED \(section 2.2.8.1.1.9\)](#) message.
 - Set the connection state to Ended.
- If the Connection referenced by the Enlistment is of type [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) or [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#):
 - If the Connection state is Modifying Timeout:
 - Send a [TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE \(section 2.2.8.1.2.8\)](#) message.
 - Otherwise, if the Connection state is Active, Aborting Transaction or Committing Transaction:
 - Send a [TXUSER_BEGIN2_MTAG_SINK_ERROR \(section 2.2.8.1.2.5\)](#)
 - The Error field MUST be set to TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED.
 - Set Connection state to Ended.

3.4.7.18 Set Transaction Attributes Failure

The Set Transaction Attributes Failure event MUST be signaled with the following arguments:

- A Transaction object.

If the Set Transaction Attributes Failure event is signaled, the transaction manager MUST perform the following actions:

- Find a [CONNTYPE_TXUSER_IMPORT2](#) connection instance in the provided transaction's connection list.
- If the connection is not found, ignore the event.
- Otherwise:
 - Send a [TXUSER_IMPORT2_MTAG_SINK_ERROR](#) message using the connection:
 - The Error field MUST be set to TRUN_TXIMPORT_ERROR_IMPORT_TX_NOT_FOUND.

- Set the connection state to Ended.

3.4.7.19 Set Transaction Attributes Success

The Set Transaction Attributes Success event MUST be signaled with the following arguments:

- A Transaction object.

If the Set Transaction Attributes Success event is signaled, the transaction manager MUST perform the following actions:

- Find a [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.2.4\)](#) connection instance in the provided transaction's connection list.
- If the connection is not found, ignore the signal.
- Otherwise:
 - Create a new Enlistment object using the following fields:
 - The Transaction Manager Communicating with an Application Facet.
 - The provided Transaction object.
 - The connection object.
 - Assign the new Enlistment object to the connection's Enlistment field.
 - Signal the [Create Voter Enlistment \(section 3.2.7.14\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the new enlistment object.

3.4.7.20 Set Transaction Timeout Failure

The Set Transaction Timeout Failure event MUST be signaled with the following arguments:

- A Transaction object.

If the Set Transaction Timeout Failure event is signaled, the transaction manager MUST perform the following actions:

- If the transaction's connection list contains a connection of type [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) or [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#):
 - Send a [TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE \(section 2.2.8.1.2.8\)](#) message using the connection.
 - Set the connection state to Active.
- Otherwise, if the transaction's connection list contains a [CONNTYPE_TXUSER_SETTXTIMEOUT \(section 2.2.8.3.3\)](#) connection:
 - Send a [TXUSER_SETTXTIMEOUT_MTAG_TOO_LATE \(section 2.2.8.1.2.8\)](#) message using the connection.
 - Set the connection state to Ended.

3.4.7.21 Set Transaction Timeout Success

The Set Transaction Timeout Success event MUST be signaled with the following arguments:

- A Transaction object.

If the Set Transaction Timeout Success event is signaled, the transaction manager MUST perform the following actions:

- If the transaction's connection list contains a connection of type [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) or [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#):
 - Send a [TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE \(section 2.2.8.1.2.6\)](#) message using the connection.
 - Set the connection state to Active.
- Otherwise, if the transaction's connection list contains a [CONNTYPE_TXUSER_SETTXTIMEOUT \(section 2.2.8.3.3\)](#) connection:
 - Send a TXUSER_SETTXTIMEOUT_MTAG_REQUEST_COMPLETE (section 2.2.8.1.2.6) message using the connection.
 - Set the connection state to Ended.

3.4.7.22 Unilaterally Aborted

The Unilaterally Aborted event MUST be signaled with the following arguments:

- An Enlistment object.

If the Unilaterally Aborted event is signaled, the transaction manager MUST perform the following actions:

- If the enlistment's connection is of type [CONNTYPE_TXUSER_BEGINNER \(section 2.2.8.1.1\)](#):
 - The connection state MUST be Active.
 - Set the connection state to Transaction Aborted.
- Otherwise, if the enlistment's connection is of type [CONNTYPE_TXUSER_BEGIN2 \(section 2.2.8.1.2\)](#) or [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#):
 - Send a [TXUSER_BEGIN2_MTAG_SINK_ERROR \(section 2.2.8.1.2.5\)](#) message:
 - The Error field MUST be set to TRUN_TXBEGIN_ERROR_NOTIFY_ABORTED.
 - Set the connection state to Ended.

3.5 Resource Manager Details

3.5.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations

adhere to this model as long as their external behavior is consistent with the behavior that is described in this document.

Note that the abstract data model can be implemented in a variety of ways. This protocol does not prescribe or advocate any specific implementation technique.

A resource manager MUST maintain all the data elements as specified in section [3.1.1](#).

A resource manager MUST also maintain the following data elements:

- **Identifier:** A durable GUID that specifies the resource manager identifier.
- **Session identifier:** A volatile GUID that specifies the **resource manager session identifier**.
- **Durable log:** A durable list of transaction objects. The contents of the log MUST persist across software restarts and transient failures.
- **Reenlistment list:** A list of connection objects.
- **Transaction manager name:** A Name object that identifies the transaction manager.
- **Reenlistment timeout:** A value that indicates the number of milliseconds the resource manager will wait for an outcome while reenlisting on a transaction.
- **Connection:** A connection object that MUST be of type [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL \(section 2.2.10.1.2\)](#) or [CONNTYPE_TXUSER_RESOURCEMANAGER \(section 2.2.10.1.1\)](#).

When a transaction object is stored in the durable log of the resource manager, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) MUST record, at minimum, the following fields:

- The **Identifier** field.

A resource manager MUST provide the states that are defined in the following sections for its supported connection types. Section [2.2.1.1.3](#) defines the connection types that a resource manager MUST provide for each supported protocol version.

3.5.1.1 CONNTYPE_TXUSER_RESOURCEMANAGER Initiator States

The resource manager MUST act as an initiator for the [CONNTYPE_TXUSER_RESOURCEMANAGER \(section 2.2.10.1.1\)](#) connection type. In this role, the resource manager MUST provide support for the following states:

- Idle
- Awaiting Create Response
- Recovering
- Awaiting Completion Confirmation
- Active
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_RESOURCEMANAGER (section 2.2.10.1.1) initiator states.

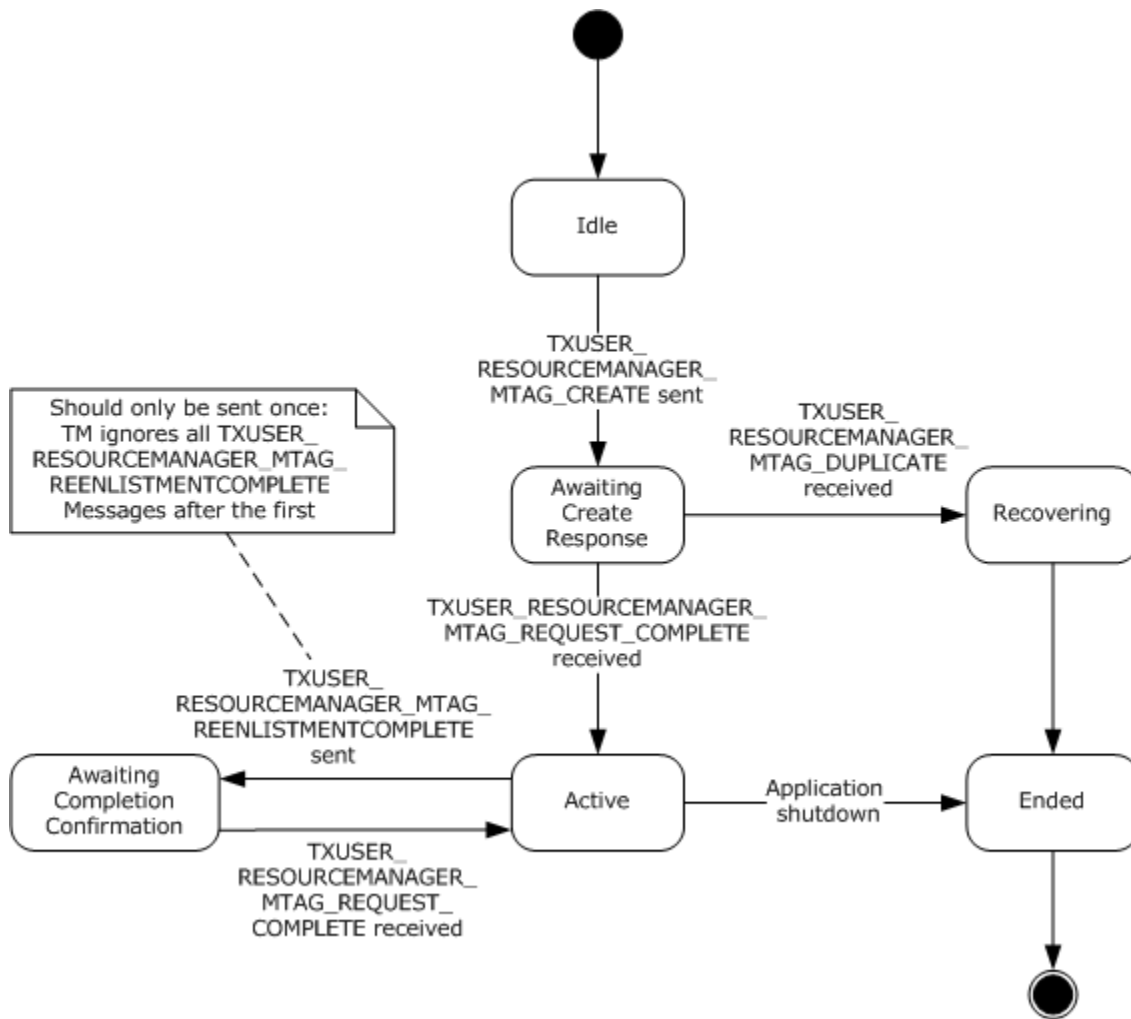


Figure 42: Resource Manager state diagram for CONNTYPE_TXUSER_RESOURCEMANAGER

3.5.1.1.1 Idle

This is the initial state. The following event is processed in this state:

- Registering with the transaction manager by using [CONNTYPE_TXUSER_RESOURCEMANAGER \(section 2.2.10.1.1\)](#).

3.5.1.1.2 Awaiting Create Response

The following events are processed in this state:

- Receiving a [TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE \(section 2.2.10.1.1.4\)](#) message.
- Receiving a [TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE \(section 2.2.10.1.1.2\)](#) message.

3.5.1.1.3 Recovering

The following event is processed in this state:

- [Reenlistment Complete \(section 3.5.7.3\)](#).

3.5.1.1.4 Awaiting Completion Confirmation

The following event is processed in this state:

- Receiving a [TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE \(section 2.2.10.1.1.4\)](#) message.

3.5.1.1.5 Active

No specific events are processed in this state.

3.5.1.1.6 Ended

This is the final state.

3.5.1.2 CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL Initiator States

The resource manager MUST act as an initiator for the [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL \(section 2.2.10.1.2\)](#) connection type. In this role, the resource manager MUST provide support for the following states:

- Idle
- Awaiting Create Response
- Recovering
- Awaiting Completion Confirmation
- Active
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL initiator states.

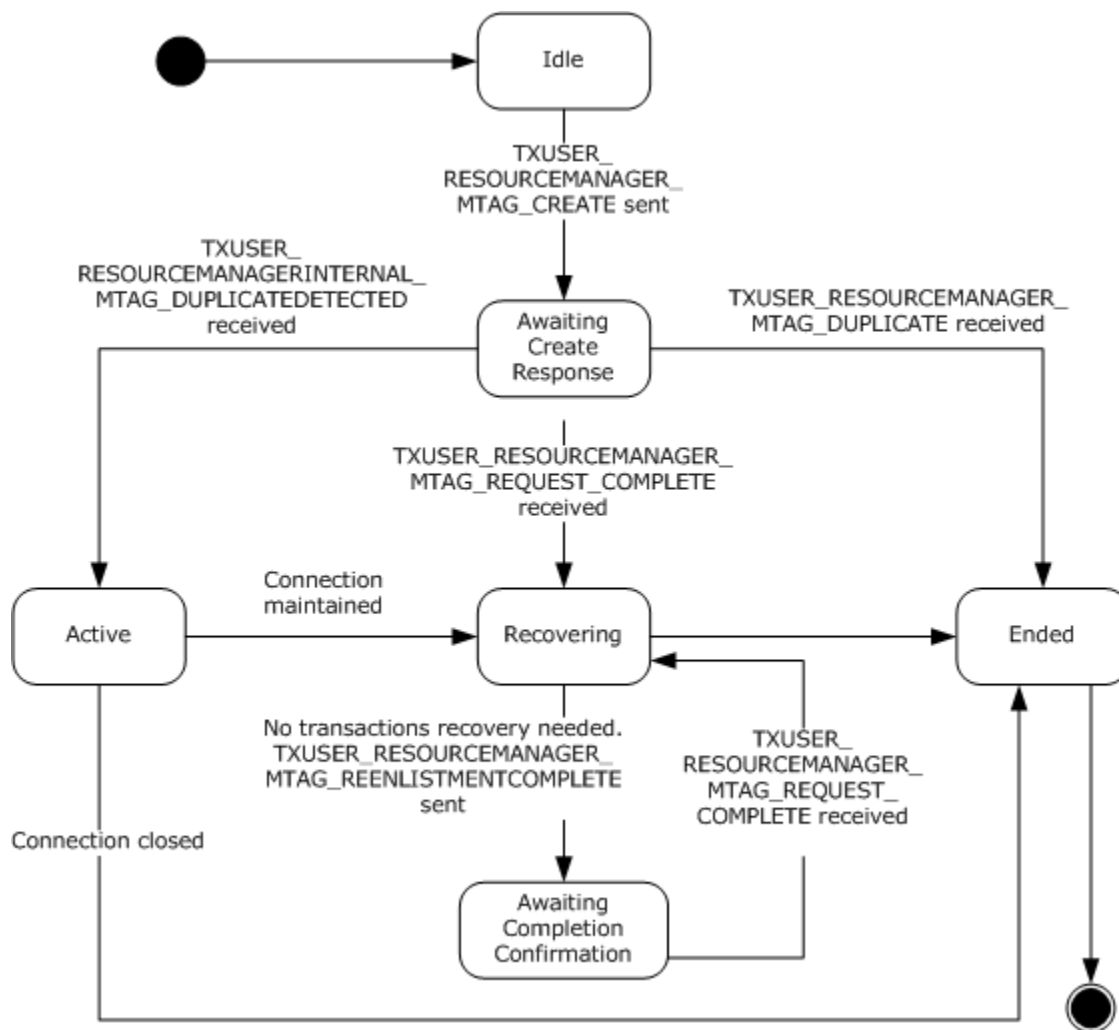


Figure 43: CONNTYPE_PARTNERTM_RESOURCEMANAGERINTERNAL Initiator States

3.5.1.2.1 Idle

This is the initial state. The following event is processed in this state:

- Registering with the transaction manager using [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL \(section 2.2.10.1.2\)](#).

3.5.1.2.2 Awaiting Create Response

The following events are processed in this state:

- Receiving a [TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE \(section 2.2.10.1.1.4\)](#) message.
- Receiving a [TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE \(section 2.2.10.1.1.2\)](#) message.

3.5.1.2.3 Recovering

The following events are processed in this state:

- [Reenlistment Complete \(section 3.5.7.3\)](#).
- Receiving [TXUSER_RESOURCEMANAGERINTERNAL MTAG_DUPLICATEDETECTED \(section 2.2.10.1.2.1\)](#) message.

3.5.1.2.4 Awaiting Completion Confirmation

The following events are processed in this state:

- Receiving [TXUSER_RESOURCEMANAGER MTAG_REQUEST_COMPLETE \(section 2.2.10.1.1.4\)](#) message.
- Receiving [TXUSER_RESOURCEMANAGERINTERNAL MTAG_DUPLICATEDETECTED \(section 2.2.10.1.2.1\)](#) message.

3.5.1.2.5 Active

The following events are processed in this state:

- Receiving [TXUSER_RESOURCEMANAGERINTERNAL MTAG_DUPLICATEDETECTED \(section 2.2.10.1.2.1\)](#) message.

3.5.1.2.6 Ended

This is the final state.

3.5.1.3 CONNTYPE_TXUSER_PHASE0 Initiator States

The resource manager MUST act as an initiator for the [CONNTYPE_TXUSER_PHASE0 \(section 2.2.10.2.1\)](#) connection type. In this role, the resource manager MUST provide support for the following states:

- Idle
- Awaiting Create Response
- Active
- Processing Phase Zero Request
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_PHASE0 initiator states.

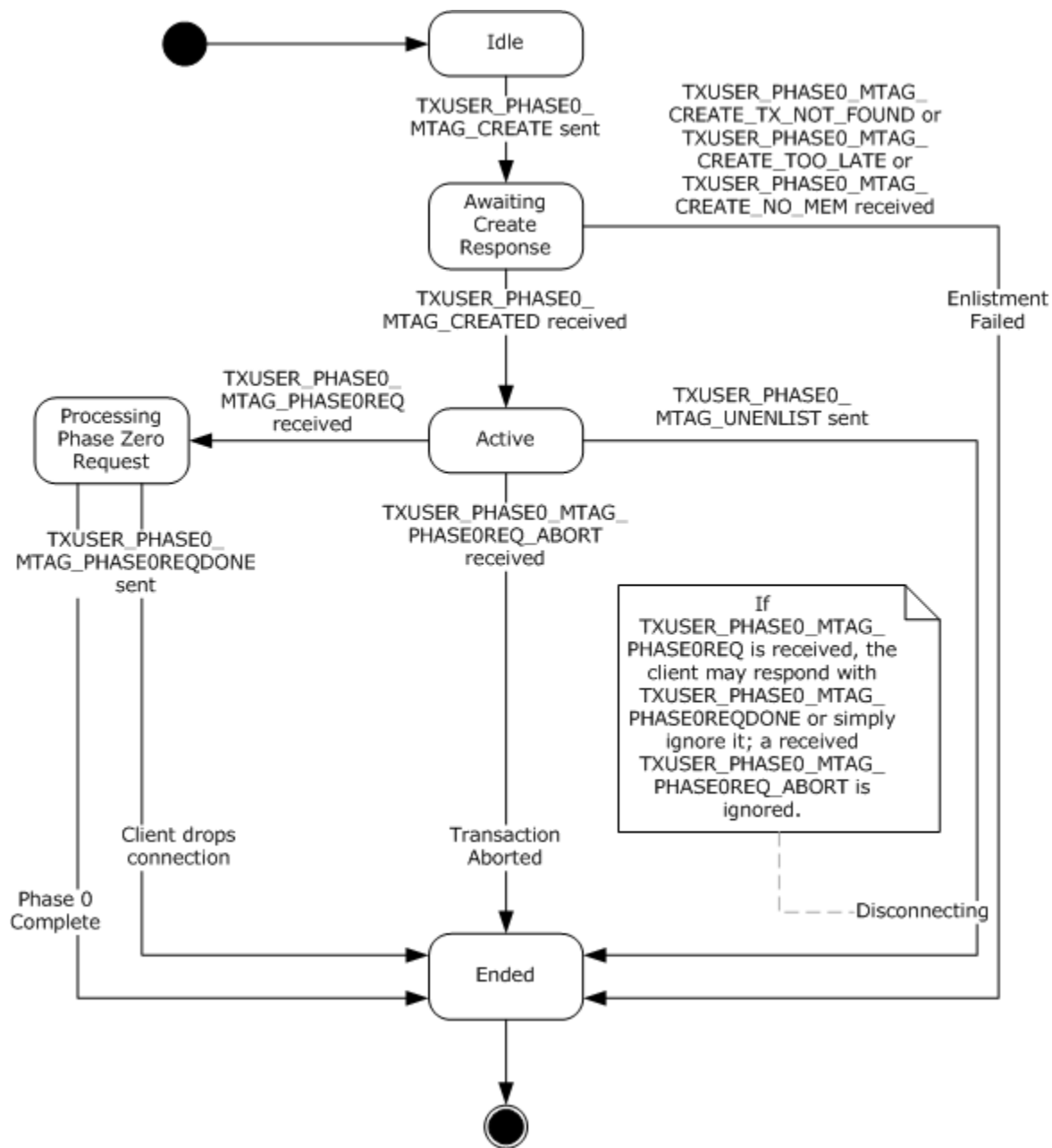


Figure 44: CONNTYPE_TXUSER_PHASE0 Initiator States

3.5.1.3.1 Idle

This is the initial state. The following event is processed in this state:

- [Enlisting as a Phase Zero Participant on a Specific Transaction \(section 3.5.4.2\).](#)

3.5.1.3.2 Awaiting Create Response

The following events are processed in this state:

- Receiving a [TXUSER PHASE0 MTAG CREATED \(section 2.2.10.2.1.4\)](#) message.
- Receiving a [TXUSER PHASE0 MTAG CREATE TX NOT FOUND \(section 2.2.10.2.1.3\)](#) or [TXUSER PHASE0 MTAG CREATE TOO LATE \(section 2.2.10.2.1.2\)](#) message.

3.5.1.3.3 Active

The following events are processed in this state:

- Receiving a [TXUSER PHASE0 MTAG PHASE0REQ \(section 3.5.5.2.1.3\)](#) message.
- Receiving a [TXUSER PHASE0 MTAG PHASE0REQ ABORT \(section 3.5.5.2.1.4\)](#) message.
- [Canceling Enlistment as a Phase Zero Participant on a Specific Transaction \(section 3.5.4.1\)](#).

3.5.1.3.4 Processing Phase 0 Request

The following event is processed in this state:

- [Phase Zero Request Completed \(section 3.5.4.8\)](#).

3.5.1.3.5 Ended

This is the final state.

3.5.1.4 CONNTYPE_TXUSER_ENLISTMENT Initiator States

The resource manager MUST act as an initiator for the [CONNTYPE_TXUSER_ENLISTMENT \(section 2.2.10.2.2\)](#) connection type. In this role, the resource manager MUST provide support for the following states:

- Idle
- Awaiting Enlistment Response
- Active
- Single Phase Committing
- Preparing For Transaction Commit
- Finalizing Abort Operations
- Awaiting Transaction Outcome
- Finalizing Commit Operations
- Ended

The following figure shows the relationship between the CONNTYPE_TXUSER_ENLISTMENT initiator states. In the figure, the parenthetical numbers are the actual enumeration values.

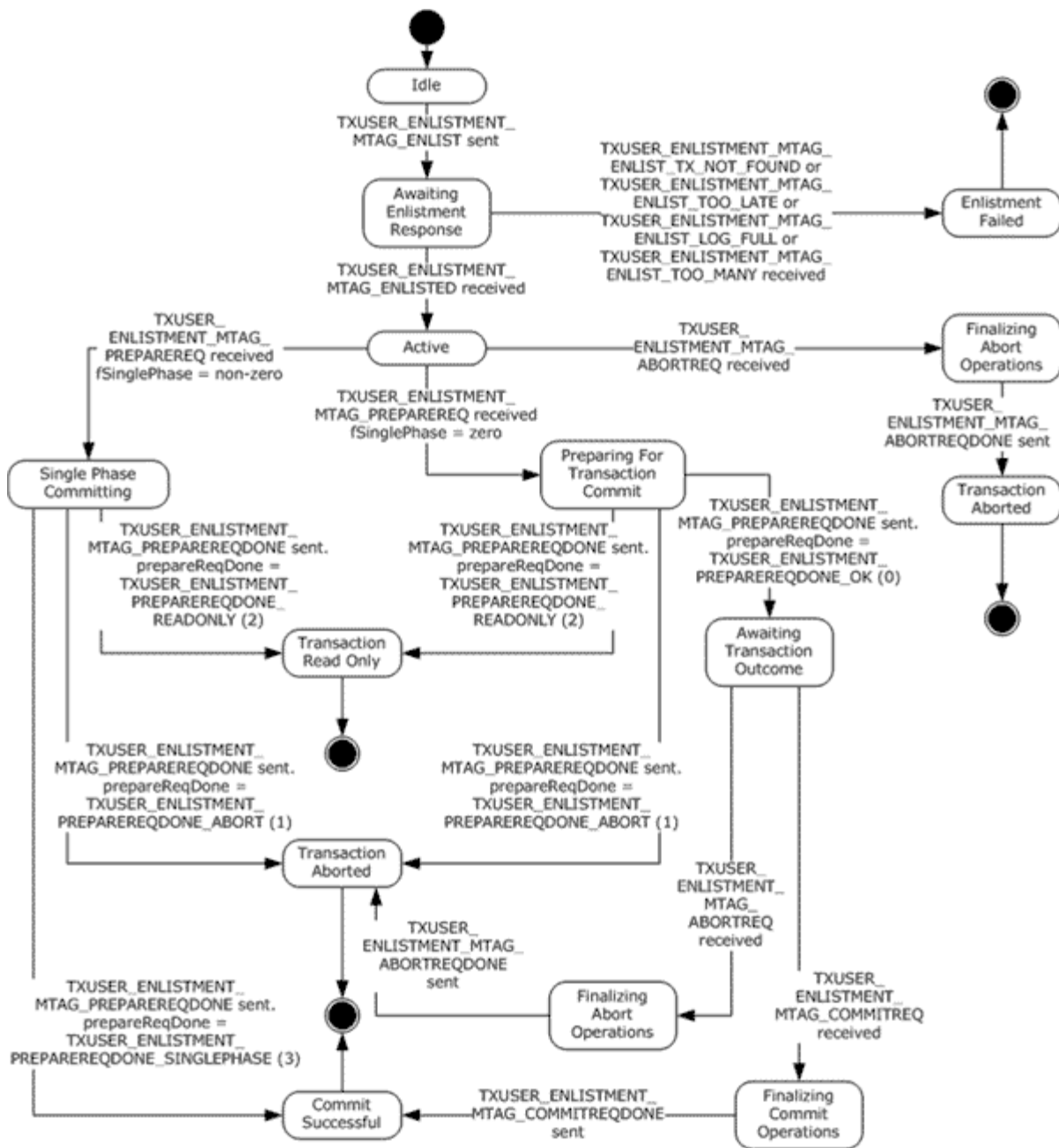


Figure 45: CONNTYPE_TXUSER_ENLISTMENT Initiator States

3.5.1.4.1 Idle

This is the initial state. The following event is processed in this state:

- [Enlisting on a Specific Transaction \(section 3.5.4.3\).](#)

3.5.1.4.2 Awaiting Enlistment Response

The following events are processed in this state:

- [Receiving a TXUSER_ENLISTMENT_MTAG_ENLISTED Message \(section 3.5.5.2.2.1\)](#),
- [Receiving a TXUSER_ENLISTMENT_MTAG_ENLIST_TX_NOT_FOUND, TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_LATE, TXUSER_ENLISTMENT_MTAG_ENLIST_LOG_FULL, or TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_MANY Message \(section 3.5.5.2.2.2\)](#)

3.5.1.4.3 Active

The following events are processed in this state:

- [Receiving a TXUSER_ENLISTMENT_MTAG_PREPAREREQ Message \(section 3.5.5.2.2.3\)](#).
- [Receiving a TXUSER_ENLISTMENT_MTAG_ABORTREQ Message \(section 3.5.5.2.2.5\)](#).

3.5.1.4.4 Single Phase Committing

The following event is processed in this state:

- [Enlistment Single-Phase Commit Request Completed \(section 3.5.4.7\)](#).

3.5.1.4.5 Preparing for Transaction Commit

The following event is processed in this state:

- [Enlistment Prepare Request Completed \(section 3.5.4.6\)](#).

3.5.1.4.6 Finalizing Commit Operations

The following event is processed in this state:

- [Enlistment Commit Request Completed \(section 3.5.4.5\)](#).

3.5.1.4.7 Ended

This is the final state.

3.5.1.5 CONNTYPE_TXUSER_REENLIST Initiator States

The resource manager MUST act as an initiator for the [CONNTYPE_TXUSER_REENLIST \(section 2.2.10.3.1\)](#) connection type. In this role, the resource manager MUST provide support for the following states:

- Idle
- Awaiting Reenlist Response
- Ended

The following figure depicts the relationship between the CONNTYPE_TXUSER_REENLIST initiator states.

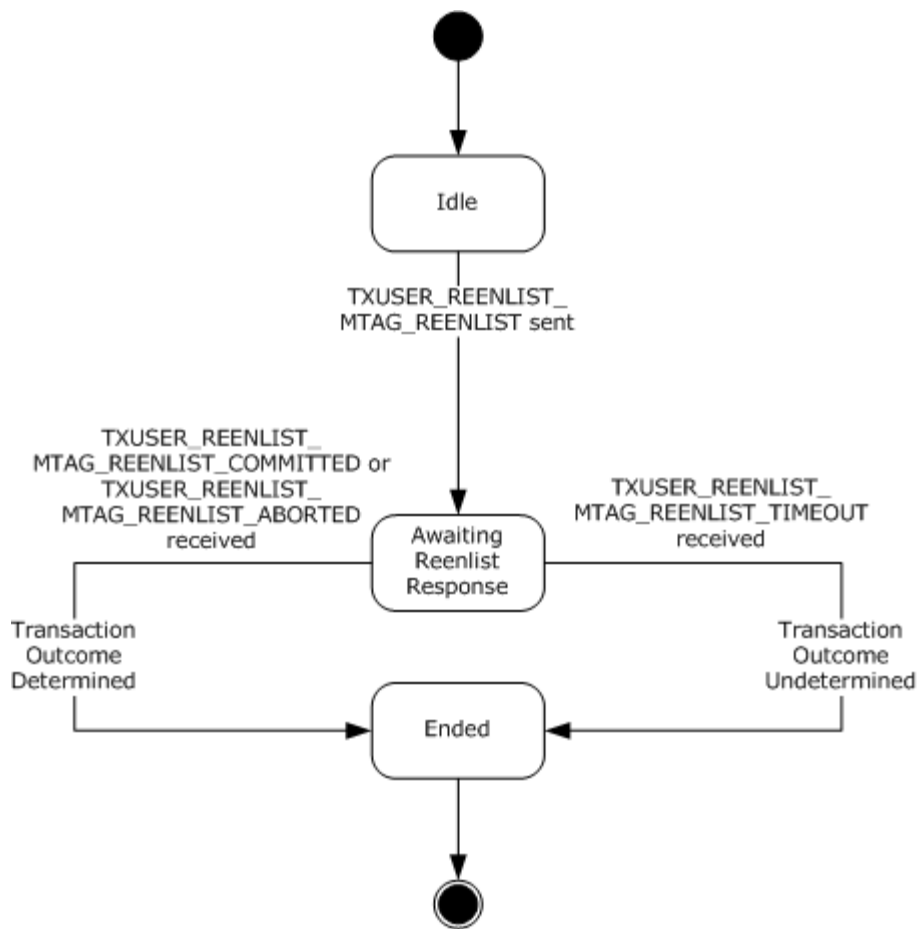


Figure 46: CONNTYPE_TXUSER_REENLIST Initiator States

3.5.1.5.1 Idle

The following events are processed in this state:

- Receiving a [TXUSER_ENLISTMENT_MTAG_COMMITREQ \(section 2.2.10.2.2.3\)](#) message.
- Receiving a [TXUSER_ENLISTMENT_MTAG_ABORTREQ \(section 2.2.10.2.2.1\)](#) message.

3.5.1.5.2 Awaiting Reenlist Response

The following event is processed in this state:

- [Enlistment Commit Request Completed \(section 3.5.4.5\)](#).

3.5.1.5.3 Ended

This is the final State.

3.5.1.6 CONNTYPE_TXUSER_VOTER Initiator States

The resource manager MUST act as an initiator for the [CONNTYPE_TXUSER_VOTER \(section 2.2.10.4.1\)](#) connection type. In this role, the resource manager MUST provide support for the following states:

- Idle
- Awaiting Creation Response
- Active
- Performing Transaction Operations
- Awaiting Outcome
- Ended

Note that the abstract data model can be implemented in a variety of ways. This protocol does not prescribe or advocate any specific implementation technique.

The following figure shows the relationship between the CONNTYPE_TXUSER_VOTER initiator states.

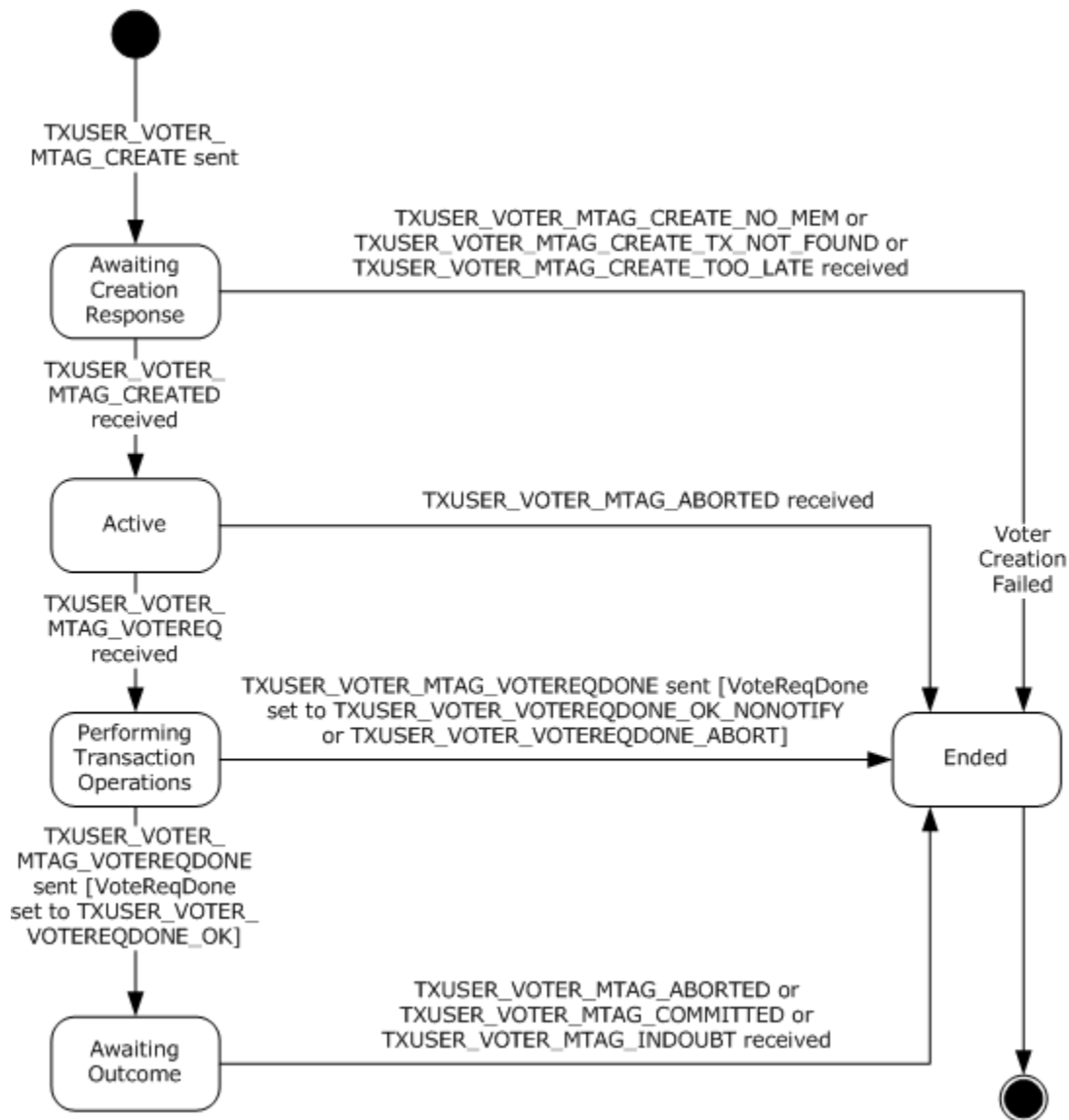


Figure 47: CONNTYPE_TXUSER_VOTER Initiator States

3.5.1.6.1 Idle

This is the initial state. The following event is processed in this state:

- [Registering as a Voter on a Specific Transaction \(section 3.5.4.9\).](#)

3.5.1.6.2 Awaiting Creation Response

The following events are processed in this state:

- Receiving a [TXUSER_VOTER_MTAG_CREATED \(section 2.2.10.4.1.7\)](#) message.

- Receiving a [TXUSER VOTER MTAG CREATE TX NOT FOUND \(section 2.2.10.4.1.6\)](#) or [TXUSER VOTER MTAG CREATE TOO LATE \(section 2.2.10.4.1.5\)](#) message.

3.5.1.6.3 Active

The following events are processed in this state:

- Receiving a [TXUSER VOTER MTAG VOTEREQ \(section 2.2.10.4.1.8\)](#) message.
- Receiving a [TXUSER STATUS MTAG ABORTED \(section 2.2.10.4.1.1\)](#) message.

3.5.1.6.4 Performing Transaction Operations

The following event is processed in this state:

- [Voter Vote Request Completed \(section 3.5.4.11\)](#).

3.5.1.6.5 Awaiting Outcome

The following events are processed in this state:

- Receiving a [TXUSER STATUS MTAG COMMITTED \(section 2.2.10.4.1.2\)](#) message.
- Receiving a [TXUSER STATUS MTAG ABORTED \(section 2.2.10.4.1.1\)](#) message.
- Receiving a [TXUSER STATUS MTAG INDOUBT \(section 2.2.10.4.1.3\)](#) message.

3.5.1.6.6 Ended

This is the final state.

3.5.2 Timers

None.

3.5.3 Initialization

When a resource manager is initialized:

- The **Identifier** field MUST be set to a GUID that is obtained from an implementation-specific source. This value MUST remain consistent across multiple software restarts or transient failures. The resource manager SHOULD create the GUID as specified in [\[RFC4122\]](#).
- The **Transaction Manager Name** field MUST be set to a value that is obtained from an implementation-specific source. This value MUST remain consistent across multiple software restarts or transient failures.
- The **Reenlistment Timeout** field MUST be set to a value that is obtained from an implementation-specific source.
- The resource manager MUST register with its transaction manager, as specified in section [3.5.4.10](#).

3.5.4 Higher-Layer Triggered Events

The resource manager operation is driven by a set of higher-layer events. These events are triggered by decisions that are made by the higher-layer business logic of the resource manager. The motivations and details of this higher-layer business logic are specific to the implementation of the resource manager and the software environment in which it executes.

The resource manager **MUST** be prepared to process the following events.

3.5.4.1 Canceling Enlistment as a Phase Zero Participant on a Specific Transaction

This event **MUST** be signaled by the higher-layer business logic with the following arguments:

- A connection object.

If the higher-layer business logic cancels its enlistment as a Phase Zero participant on a specific transaction, the resource manager **MUST** perform the following steps:

- Send a [TXUSER PHASE0 MTAG UNENLIST \(section 2.2.10.2.1.8\)](#) message using the connection.
- Set the connection state to [Ended \(section 3.2.1.4.14\)](#).

3.5.4.2 Enlisting as a Phase Zero Participant on a Specific Transaction

This event **MUST** be signaled by the higher-layer business logic with the following arguments:

- A transaction object.

If the higher-layer business logic enlists as a Phase Zero participant on a specific transaction, the resource manager **MUST** perform the following steps:

- If the transaction manager of the resource manager supports the [CONNTYPE_TXUSER_PHASE0](#) connection type, as specified in section [2.2.10.2.1](#):
 - Initiate a new CONNTYPE_TXUSER_PHASE0 (section 2.2.10.2.1) connection to the transaction manager.
 - Send a [TXUSER PHASE0 MTAG CREATE \(section 2.2.10.2.1.1\)](#) message using the connection:
 - Set the **guidTx** field to the transaction identifier.
 - Set the connection state to Awaiting Enlistment Response.
- Otherwise, the resource manager **MUST** return a Failure to the higher-layer business logic.

3.5.4.3 Enlisting on a Specific Transaction

This event **MUST** be signaled by the higher-layer business logic with the following arguments:

- A transaction object.

If the higher-layer business logic decides to enlist on a specific transaction, the resource manager **MUST** perform the following steps:

- Initiate a new [CONNTYPE_TXUSER_ENLISTMENT \(section 2.2.10.2.2\)](#) connection to the transaction manager.
- Assign the transaction object to the connection-specific data of the connection.
- Add the connection to the connection list of the transaction.
- Send a [TXUSER_ENLISTMENT_MTAG_ENLIST \(section 2.2.10.2.2.5\)](#) message using the connection:
 - Set the **guidTX** field to the identifier of the transaction object.
 - Set the **guidTM** field to the Identifier field of the resource manager.
 - Set the **guidSession** field to the Session Identifier field of the resource manager.
- Set the connection state to Awaiting Enlistment Response.

3.5.4.4 Enlistment Abort Request Completed

This event MUST be signaled by the higher-layer business logic with the following arguments:

- A transaction object.

When the higher-layer business logic completes an enlistment Abort request, as specified in section [3.5.5.2.2.5](#), the resource manager MUST perform the following steps:

- Find a [CONNTYPE_TXUSER_ENLISTMENT \(section 2.2.10.2.2\)](#) connection in the connection list of the transaction object.
- If the connection is found:
 - If the transaction object was added to the durable log:
 - Remove the transaction object from the durable log.
 - Send a [TXUSER_ENLISTMENT_MTAG_ABORTREQDONE \(section 2.2.10.2.2.2\)](#) message using the connection.
 - Set the connection state to Ended.
- Otherwise, find a [CONNTYPE_TXUSER_REENLIST \(section 2.2.10.3.1\)](#) connection in the connection list of the transaction object.
- If the connection is found:
 - If the transaction object was added to the durable log:
 - Remove the transaction object from the durable log.
 - Remove the connection from the reenlistment list of the resource manager.
 - If the list is now empty:
 - Signal the [Reenlistment Complete \(section 3.5.7.3\)](#) event on the resource manager.
 - Set the connection state to Ended.
- Otherwise, ignore the event.

3.5.4.5 Enlistment Commit Request Completed

This event MUST be signaled by the higher-layer business logic with the following arguments:

- A transaction object.

When the higher-layer business logic completes an enlistment Commit request as specified in section [3.5.5.2.2.3](#), the resource manager MUST perform the following steps:

- Find a [CONNTYPE_TXUSER_ENLISTMENT \(section 2.2.10.2.2\)](#) connection in the connection list of the transaction object.
- If the connection is found:
 - Remove the transaction object from the durable log.
 - Send a [TXUSER_ENLISTMENT_MTAG_COMMITREQDONE \(section 2.2.10.2.2.4\)](#) message using the connection.
 - Set the connection state to Ended.
- Otherwise, find a [CONNTYPE_TXUSER_REENLIST \(section 2.2.10.3.1\)](#) connection in the connection list of the transaction object.
- If the connection is found:
 - Remove the transaction object from the durable log.
 - Remove the connection from the reenlistment list of the resource manager.
 - If the list is now empty:
 - Signal the [Reenlistment Complete \(section 3.5.7.3\)](#) event on the resource manager.
 - Set the connection state to Ended.
- Otherwise, ignore the event.

3.5.4.6 Enlistment Prepare Request Completed

This event MUST be signaled by the higher-layer business logic with the following arguments:

- A transaction object.
- An outcome value. This value MUST be one of the following:
 - Prepared
 - Read Only
 - Aborted

When the higher-layer business logic completes a Prepare request, as specified in section [3.5.1.4](#), the resource manager MUST perform the following steps:

- Obtain the [CONNTYPE_TXUSER_ENLISTMENT \(section 2.2.10.2.2\)](#) connection from the connection list of the transaction object.
- If the request outcome is Prepared:

- Add the transaction object to the durable log.
- Send a [TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE \(section 2.2.10.2.2.12\)](#) message using the connection:
 - Set the **prepareReqDone** field to TXUSER_ENLISTMENT_PREPAREREQDONE_OK.
- Set the connection state to Awaiting transaction outcome.
- Otherwise, if the request outcome is Read-Only:
 - Send a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE message using the connection:
 - Set the **prepareReqDone** field to TXUSER_ENLISTMENT_PREPAREREQDONE_READ_ONLY.
 - Set the connection state to Ended.
- Otherwise, if the request outcome is Aborted:
 - Send a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE message using the connection:
 - Set the **prepareReqDone** field to TXUSER_ENLISTMENT_PREPAREREQDONE_ABORT.
 - Set the connection state to Ended.

3.5.4.7 Enlistment Single-Phase Commit Request Completed

This event MUST be signaled by the higher-layer business logic with the following arguments:

- A transaction object.
- An outcome value. This value MUST be one of the following:
 - Prepared
 - Committed
 - Aborted

When the higher-layer business logic completes an Enlistment Single-Phase Commit request as specified in section [3.5.1.4](#), the resource manager MUST perform the following steps:

- Obtain the [CONNTYPE_TXUSER_ENLISTMENT](#) connection from the connection list of the transaction object.
- If the request outcome is Prepared:
 - Add the transaction object to the durable log.
 - Send a [TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE](#) message using the connection:
 - Set the **prepareReqDone** field to TXUSER_ENLISTMENT_PREPAREREQDONE_OK.
 - Set the connection state to Awaiting Transaction outcome.
- Otherwise, if the request outcome is Committed:
 - Send a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE message using the connection:

- Set the **prepareReqDone** field to TXUSER_ENLISTMENT_PREPAREREQDONE_SINGLEPHASE.
- Set the connection state to Ended.
- Otherwise, if the request outcome is Aborted:
 - Send a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE message using the connection:
 - Set the **prepareReqDone** field to TXUSER_ENLISTMENT_PREPAREREQDONE_ABORT.
 - Set the connection state to Ended.

3.5.4.8 Phase Zero Request Completed

This event MUST be signaled by the higher-layer business logic with the following arguments:

- A connection object.
- An outcome value. This value MUST be one of the following:
 - Read Only
 - Aborted

When the higher-layer business logic completes a [Phase Zero](#) request, the resource manager MUST perform the following steps:

- If the Phase Zero outcome is Read Only:
 - Send a [TXUSER PHASE0_MTAG_PHASE0REQDONE](#) message.
 - Set the connection state to [Ended](#).
- Otherwise, if the Phase Zero outcome is Aborted:
 - Set the connection state to Ended.

3.5.4.9 Registering as a Voter on a Specific Transaction

This event MUST be signaled by the higher-layer business logic with the following arguments:

- A transaction object.

If the higher-layer business logic decides to register as a voter on a specific transaction manager, the resource manager MUST perform the following steps:

- Initiate a new [CONNTYPE TXUSER VOTER \(section 2.2.10.4.1\)](#) connection to the transaction manager.
- Send a [TXUSER VOTER_MTAG_CREATE \(section 2.2.10.4.1.4\)](#) message using the connection:
 - Set the **guidTX** field to the transaction identifier.
- Set the connection state to Awaiting Creation Response.

3.5.4.10 Registering with Transaction Manager

If the higher-layer business logic wants to register with the transaction manager, the resource manager MUST perform the following actions:

- The resource manager SHOULD set the session identifier field to a new GUID value as specified in [\[RFC4122\]](#). Optionally, the resource manager MAY [<49>](#) instead set the session identifier field to NULL GUID.
- If the transaction manager's resource manager supports the [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL](#) connection type as specified in section [2.2.1.1.3](#):
 - The resource manager MUST attempt to [register with the transaction manager](#) using CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL.
- Otherwise:
 - The resource manager MUST attempt to [register with the transaction manager](#) using [CONNTYPE_TXUSER_RESOURCEMANAGER](#).

3.5.4.10.1 Registering with Transaction Manager Using CONNTYPE_TXUSER_RESOURCEMANAGER

The resource manager MUST perform the following actions:

- Initiate a new [CONNTYPE_TXUSER_RESOURCEMANAGER](#) connection to the provided Name object.
- Assign the new connection to the Connection field of the resource manager.
- Send a [TXUSER_RESOURCEMANAGER_MTAG_CREATE](#) message using the connection:
 - Set the **guidRM** field to the identifier of the resource manager.
 - Set the **guidSession** field to the session identifier of the resource manager.
- Set the connection state to Awaiting Create Response.

3.5.4.10.2 Registering with Transaction Manager Using CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL

The resource manager MUST perform the following actions:

- Initiate a new [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL](#) connection using the transaction manager Name field of the resource manager.
- Assign the new connection to the Connection field of the resource manager.
- Send a [TXUSER_RESOURCEMANAGER_MTAG_CREATE](#) message using the connection:
 - Set the **guidRM** field to the Identifier field of the resource manager.
 - Set the **guidSession** field to the Session Identifier field of the resource manager.
- Set the connection state to Awaiting Create Response.

3.5.4.11 Voter Vote Request Completed

This event MUST be signaled by the higher-layer business logic with the following arguments:

- A connection object.
- An outcome value. This value MUST be one of the following:
 - Prepared
 - Read-Only
 - Aborted

When the higher-layer business logic completes a Voter Vote request, the resource manager MUST perform the following steps:

- If the vote outcome is Prepared:
 - Send a [TXUSER_VOTER_MTAG_VOTEREQDONE](#) message using the connection:
 - Set the **VoteReqDone** field to TXUSER_VOTER_VOTEREQDONE_OK.
 - Set the connection state to Awaiting Outcome.
- Otherwise, if the vote outcome is Read Only:
 - Send a TXUSER_VOTER_MTAG_VOTEREQDONE message using the connection:
 - Set the **VoteReqDone** field to TXUSER_VOTER_VOTEREQDONE_OK_NONOTIFY.
 - Set the connection state to Ended.
- Otherwise, if the vote outcome is Aborted:
 - Send a TXUSER_VOTER_MTAG_VOTEREQDONE message using the connection:
 - Set the **VoteReqDone** field to TXUSER_VOTER_VOTEREQDONE_ABORT.
 - Set the connection state to Ended.

3.5.5 Message Processing Events and Sequencing Rules

3.5.5.1 Resource Manager Registration

3.5.5.1.1 CONNTYPE_TXUSER_RESOURCEMANAGER as Initiator

For all messages that are received in this connection type, the application MUST process the messages as specified in section [3.1](#). The application MUST additionally follow the processing rules as specified in the following sections.

3.5.5.1.1.1 Receiving a TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE Message

When the resource manager receives a [TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Create Response:
 - Set the connection state to Ended.
 - Return a failure result to the higher-layer business logic.

3.5.5.1.1.2 Receiving a TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE Message

When the resource manager receives a [TXUSER_RESOURCEMANAGER_REQUEST_COMPLETE](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Create Response:
 - Set the connection state to Recovering.
 - Signal the Recover Transactions event on the resource manager.
- Otherwise, if the connection state is Awaiting Completion Confirmation:
 - Set the connection state to Active
 - Return a success result to the higher-layer business logic.

3.5.5.1.1.3 Connection Disconnected

When a [CONNTYPE_TXUSER_RESOURCEMANAGER \(section 2.2.10.1.1\)](#) connection is disconnected, the resource manager MUST perform the following actions:

- If the connection state is Active, Awaiting Create Response, Recovering, or Awaiting Completion Confirmation:
 - Signal the [Transaction Manager Down \(section 3.5.7.4\)](#) event on the resource manager.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.5.5.1.2 CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL as Initiator

For all messages received in this connection type, the application MUST process the messages as specified in section [3.1](#). The application MUST additionally follow the processing rules as specified in the following sections.

3.5.5.1.2.1 Receiving a TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE or TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE Message

When the resource manager receives either the [TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE](#) or [TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE](#) message, it MUST follow the same message processing rules as the [CONNTYPE_TXUSER_RESOURCEMANAGER](#) connection type when it acts as the initiator. See section [3.5.5.1.1](#) for more information

3.5.5.1.2.2 Receiving a TXUSER_RESOURCEMANAGERINTERNAL_MTAG_DUPLICATEDETECTED Message

When the resource manager receives a [TXUSER_RESOURCEMANAGERINTERNAL_MTAG_DUPLICATEDETECTED](#) message, the resource manager MUST perform the following actions:

- If the connection state is Recovering, Awaiting Completion Confirmation, or Active:
 - Inform the higher-layer business logic that the transaction manager has detected a duplicate resource manager registration.

3.5.5.1.2.3 Connection Disconnected

When a [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL](#) (section 2.2.10.1.2) connection is disconnected, the event MUST be processed as specified in section [3.1.8.1](#).

3.5.5.2 Transaction Coordination

3.5.5.2.1 CONNTYPE_TXUSER_PHASE0 as Initiator

For all messages that are received in this connection type, the resource manager MUST process the message as specified in section [3.1](#). The resource manager MUST additionally follow the processing rules as specified in the following sections.

3.5.5.2.1.1 Receiving a TXUSER_PHASE0_MTAG_CREATED Message

When the resource manager receives a [TXUSER_PHASE0_MTAG_CREATED](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Create Response:
 - Set the connection state to Active.
 - Return a success result to the higher-layer business logic.
 - Add the connection to the connection list of the transaction.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.2.1.2 Receiving a TXUSER_PHASE0_MTAG_CREATE_TX_NOT_FOUND or TXUSER_PHASE0_MTAG_CREATE_TOO_LATE Message

When the resource manager receives either the [TXUSER_PHASE0_MTAG_CREATE_TOO_LATE](#) or [TXUSER_PHASE0_MTAG_CREATE_TX_NOT_FOUND](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Create Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.2.1.3 Receiving a TXUSER_PHASE0_MTAG_PHASE0REQ Message

When the resource manager receives a [TXUSER_PHASE0_MTAG_PHASE0REQ](#) message, the resource manager MUST perform the following actions:

- If the connection state is Active:
 - Set the connection state to Processing Phase Zero Request.

- Send a [Phase Zero](#) request to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.2.1.4 Receiving a TXUSER_PHASE0_MTAG_PHASE0REQ_ABORT Message

When the resource manager receives a [TXUSER_PHASE0_MTAG_PHASE0REQ_ABORT](#) message, the resource manager MUST perform the following actions:

- If the connection state is Active:
 - Send a Transaction Aborted notification to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.2.1.5 Connection Disconnected

When a [CONNTYPE_TXUSER_PHASE0 \(section 2.2.10.2.1\)](#) connection is disconnected, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Create Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, if the connection state is Active or Processing Phase Zero Request:
 - Send a Transaction Aborted notification to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.5.5.2.2 CONNTYPE_TXUSER_ENLISTMENT as Initiator

For all messages that are received in this connection type, the resource manager MUST process the message as specified in section [3.1](#). The resource manager MUST additionally follow the processing rules as specified in the following sections.

3.5.5.2.2.1 Receiving a TXUSER_ENLISTMENT_MTAG_ENLISTED Message

When the resource manager receives a [TXUSER_ENLISTMENT_MTAG_ENLISTED](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Enlistment Response:
 - Set the connection state to Active.
 - Return a success result to the higher-layer business logic.
 - Add the connection to the connection list.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.2.2.2 Receiving a TXUSER_ENLISTMENT_MTAG_ENLIST_TX_NOT_FOUND, TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_LATE, TXUSER_ENLISTMENT_MTAG_ENLIST_LOG_FULL, or TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_MANY Message

When the resource manager receives a [TXUSER_ENLISTMENT_MTAG_ENLIST_TX_NOT_FOUND](#), [TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_LATE](#), [TXUSER_ENLISTMENT_MTAG_ENLIST_LOG_FULL](#), or [TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_MANY](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Enlistment Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.5](#).

3.5.5.2.2.3 Receiving a TXUSER_ENLISTMENT_MTAG_PREPAREREQ Message

When the resource manager receives a [TXUSER_ENLISTMENT_MTAG_PREPAREREQ](#) message, the resource manager MUST perform the following actions:

- If the connection state is Active:
 - If the fSinglePhase field of the message is non-zero:
 - Set the connection state to Single-Phase Committing.
 - Send a Single-Phase Commit request to the higher-layer business logic.
 - Otherwise:
 - Set the connection state to Preparing For Transaction Commit.
 - Send a Prepare request to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.5](#).

3.5.5.2.2.4 Receiving a TXUSER_ENLISTMENT_MTAG_COMMITREQ Message

When the resource manager receives a [TXUSER_ENLISTMENT_MTAG_COMMITREQ](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Transaction Outcome:
 - Set the connection state to Finalizing Commit Operations.
 - Send a Commit Request to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.2.2.5 Receiving a TXUSER_ENLISTMENT_MTAG_ABORTREQ Message

When the resource manager receives a [TXUSER_ENLISTMENT_MTAG_ABORTREQ](#) message, the resource manager MUST perform the following actions:

- If the connection state is Active:
 - Send an Abort request to the higher-layer business logic.
 - Set the connection state to Finalizing Abort Operations.
- Otherwise, if the connection state is Awaiting Transaction Outcome:
 - Remove the transaction object from the durable log.
 - Send an Abort request to the higher-layer business logic.
 - Set the connection state to Finalizing Abort Operations.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.5](#).

3.5.5.2.2.6 Connection Disconnected

When a [CONNTYPE_TXUSER_ENLISTMENT \(section 2.2.10.2.2\)](#) connection is disconnected, the resource manager MUST perform the following actions:

- If the connection state is either Awaiting Enlistment Response, Active, or Preparing For Transaction Commit:
 - Send an Abort request to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.5.5.3 Transaction Recovery

3.5.5.3.1 CONNTYPE_TXUSER_REENLIST as Initiator

For all messages that are received in this connection type, the resource manager MUST process the message as specified in section [3.1](#). The resource manager MUST additionally follow the processing rules as specified in the following sections.

3.5.5.3.1.1 Receiving a TXUSER_REENLIST_MTAG_COMMITTED Message

When the resource manager receives a [TXUSER_REENLIST_MTAG_COMMITTED](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Reenlist Response:
 - Send a Commit request to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.5](#).

3.5.5.3.1.2 Receiving a TXUSER_REENLIST_MTAG_ABORTED Message

When the resource manager receives a [TXUSER_REENLIST_MTAG_REENLIST_ABORTED](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Reenlist Response:
 - Send an Abort request to the higher-layer business logic.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.3.1.3 Receiving a TXUSER_REENLIST_MTAG_REENLIST_TIMEOUT Message

When the resource manager receives a [TXUSER_REENLIST_MTAG_REENLIST_TIMEOUT](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Request Response:
 - Signal the [Recover Transaction](#) event.
- Otherwise, the message MUST be processed as an invalid message as specified in section .

3.5.5.3.1.4 Connection Disconnected

This event MUST be processed as specified in section [3.1.8.1](#).

3.5.5.4 Voting

3.5.5.4.1 CONNTYPE_TXUSER_VOTER as Initiator

For all messages that are received in this connection type, the resource manager MUST process the message as specified in section [3.1](#). The resource manager MUST additionally follow the processing rules as specified in the following sections.

3.5.5.4.1.1 Receiving a TXUSER_VOTER_MTAG_CREATED Message

When the resource manager receives a [TXUSER_VOTER_MTAG_CREATED](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Creation Response:
 - Return a success result to the higher-layer business logic.
 - Set the connection state to Active.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.4.1.2 Receiving a TXUSER_VOTER_MTAG_CREATE_TX_NOT_FOUND or TXUSER_VOTER_MTAG_CREATE_TOO_LATE Message

When the resource manager receives either a [TXUSER_VOTER_MTAG_CREATE_TX_NOT_FOUND](#) or [TXUSER_VOTER_MTAG_CREATE_TOO_LATE](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Creation Response:
 - Return a failure result to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.4.1.3 Receiving a TXUSER_VOTER_MTAG_VOTEREQ Message

When the resource manager receives a [TXUSER_VOTER_MTAG_VOTEREQ](#) message, the resource manager MUST perform the following actions:

- If the connection state is Active:
 - Send a Vote request to the higher-layer business logic.
 - Set the connection state to Performing Transaction Operations.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.4.1.4 Receiving a TXUSER_STATUS_MTAG_COMMITTED Message

When the resource manager receives a [TXUSER_STATUS_MTAG_COMMITTED](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Outcome:
 - Send a Transaction Committed notification to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.4.1.5 Receiving a TXUSER_STATUS_MTAG_ABORTED Message

When the resource manager receives a [TXUSER_STATUS_MTAG_ABORTED](#) message, the resource manager MUST perform the following actions:

- If the connection state is Active or Awaiting Outcome:
 - Send a Transaction Aborted notification to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.4.1.6 Receiving a TXUSER_STATUS_MTAG_INDOUBT Message

When the resource manager receives a [TXUSER_STATUS_MTAG_INDOUBT](#) message, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Outcome:
 - Send a Transaction [In Doubt](#) notification to the higher-layer business logic.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.5.5.4.1.7 Connection Disconnected

When a [CONNTYPE_TXUSER_VOTER \(section 2.2.10.4.1\)](#) connection is disconnected, the resource manager MUST perform the following actions:

- If the connection state is Awaiting Creation Response:
 - Return a failure result to the higher-layer business logic.
- Otherwise, if the connection state is Awaiting Outcome:

- Send a Transaction [In Doubt](#) notification to the higher-layer business logic.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.5.6 Timer Events

No timer events.

3.5.7 Other Local Events

The resource manager MUST be prepared to process the local events that appear in this section.

3.5.7.1 Recover Transaction

The **Recover Transaction** event MUST be signaled with the following arguments:

- A transaction object.

If the **Recover Transaction** event is signaled, the resource manager MUST perform the following steps:

- Initiate a new [CONNTYPE TXUSER REENLIST \(section 2.2.10.3.1\)](#) connection to the transaction manager, using the Transaction Manager Name field of the resource manager.
- Add the connection to the reenlistment list of the resource manager.
- Add the connection to the connection list of the transaction object.
- Send a [TXUSER REENLIST MTAG REENLIST \(section 2.2.10.3.1.1\)](#) message using the connection:
 - Set the **guidTx** field to the Identifier field of the transaction.
 - Set the **ulTimeout** field to the Reenlistment Timeout field of the resource manager.
 - Set the **guidRm** field to the Identifier field of the resource manager.
- Set the connection state to Awaiting Reenlist Response.

3.5.7.2 Recover Transactions

If the **Recover Transactions** event is signaled, the resource manager MUST perform the following steps:

- If the durable log of the resource manager is empty:
 - Signal the [Reenlistment Complete \(section 3.5.7.3\)](#) event on the transaction manager communicating with a resource manager facet.
- Otherwise, for each transaction object in the durable log:
 - Signal the [Recover Transaction \(section 3.5.7.1\)](#) event on the transaction manager communicating with a resource manager facet with the transaction object.

3.5.7.3 Reenlistment Complete

If the **Reenlistment Complete** event is signaled, the resource manager MUST perform the following actions:

- Send a [TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE](#) message using the connection that is referenced by the Connection field of the resource manager.
- Set the connection state to Awaiting Completion Confirmation.

3.5.7.4 Transaction Manager Down

When the **Transaction Manager Down** event is signaled, the resource manager MUST perform the following steps:

- If the OleTx session that is established with the transaction manager still contains active connections in the connection list:
 - Wait for those connections to be disconnected.
- Inform the higher-layer business logic that the transaction manager has disconnected.
- The higher-layer business MUST request that the resource manager reregister with the transaction manager. The timing of the request is implementation specific.

3.6 Transaction Manager Communicating with Resource Manager Facet Details

3.6.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model if their external behavior is consistent with the behavior that is described in this document.

Note that the abstract data model can be implemented in a variety of ways. This protocol does not prescribe or advocate any specific implementation technique.

The transaction manager communicating with a resource manager facet MUST maintain all the data elements as specified in section [3.2.1.3](#).

The transaction manager communicating with a resource manager facet MUST additionally maintain the following data elements:

- **Active Resource Manager Table:** A table of entries to resource manager objects, keyed by resource manager identifier.
- **Failed to Notify List:** A list of Enlistment objects representing remote resource managers that have not yet acknowledged the commit outcome of a transaction.

A resource manager object MUST contain the following data structures:

- **Identifier:** Specifies the resource manager identifier.
- **Session Identifier:** A GUID that specifies the resource manager session identifier.

- **Connection:** The [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL \(section 2.2.10.1.2\)](#) or [CONNTYPE_TXUSER_RESOURCEMANAGER \(section 2.2.10.1.1\)](#) connection object that is associated with the resource manager.

Enlistment objects that are created by the transaction manager communicating with a resource manager facet MUST provide the following properties as specified in section [3.2.1.3](#):

- **Name:** The resource manager identifier field of the Enlistment object, formatted as a string as specified in [\[C706\]](#) Appendix A.
- **Identifier:** An empty string.

The transaction manager communicating with a resource manager MUST provide the states as specified in the following sections for its supported connection types. Section [2.2.1.1.3](#) defines the connection types that a transaction manager communicating with a resource manager MUST provide for each supported protocol version.

3.6.1.1 CONNTYPE_TXUSER_RESOURCEMANAGER Acceptor States

The transaction manager communicating with a resource manager MUST act as an acceptor for the [CONNTYPE_TXUSER_RESOURCEMANAGER \(section 2.2.10.1.1\)](#) connection type. In this role, the transaction manager communicating with a resource manager MUST provide support for the following states:

- [Idle \(section 3.6.1.1.1\)](#)
- [Creating \(section 3.6.1.1.2\)](#)
- [Reenlisting \(section 3.6.1.1.3\)](#)
- [Active \(section 3.6.1.1.4\)](#)
- [Ended \(section 3.6.1.1.5\)](#)

3.6.1.1.1 Idle

The Idle state is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_RESOURCEMANAGER_MTAG_CREATE Message \(section 3.6.5.1.1.1\)](#).

3.6.1.1.2 Creating

The following events are processed in the Creating state:

- [Create Resource Manager \(section 3.6.7.9\)](#).

3.6.1.1.3 Reenlisting

The following events are processed in the Reenlisting state:

- [Receiving a TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE Message \(section 3.6.5.1.1.2\)](#).

3.6.1.1.4 Active

The following events are processed in the Active state:

- [Reenlist Complete \(section 3.6.7.15\).](#)

3.6.1.1.5 Ended

The final state is the Ended state.

3.6.1.1.6 State Diagram

The following figure shows the relationship between the [CONNTYPE_TXUSER_RESOURCEMANAGER \(section 2.2.10.1.1\)](#) acceptor states.

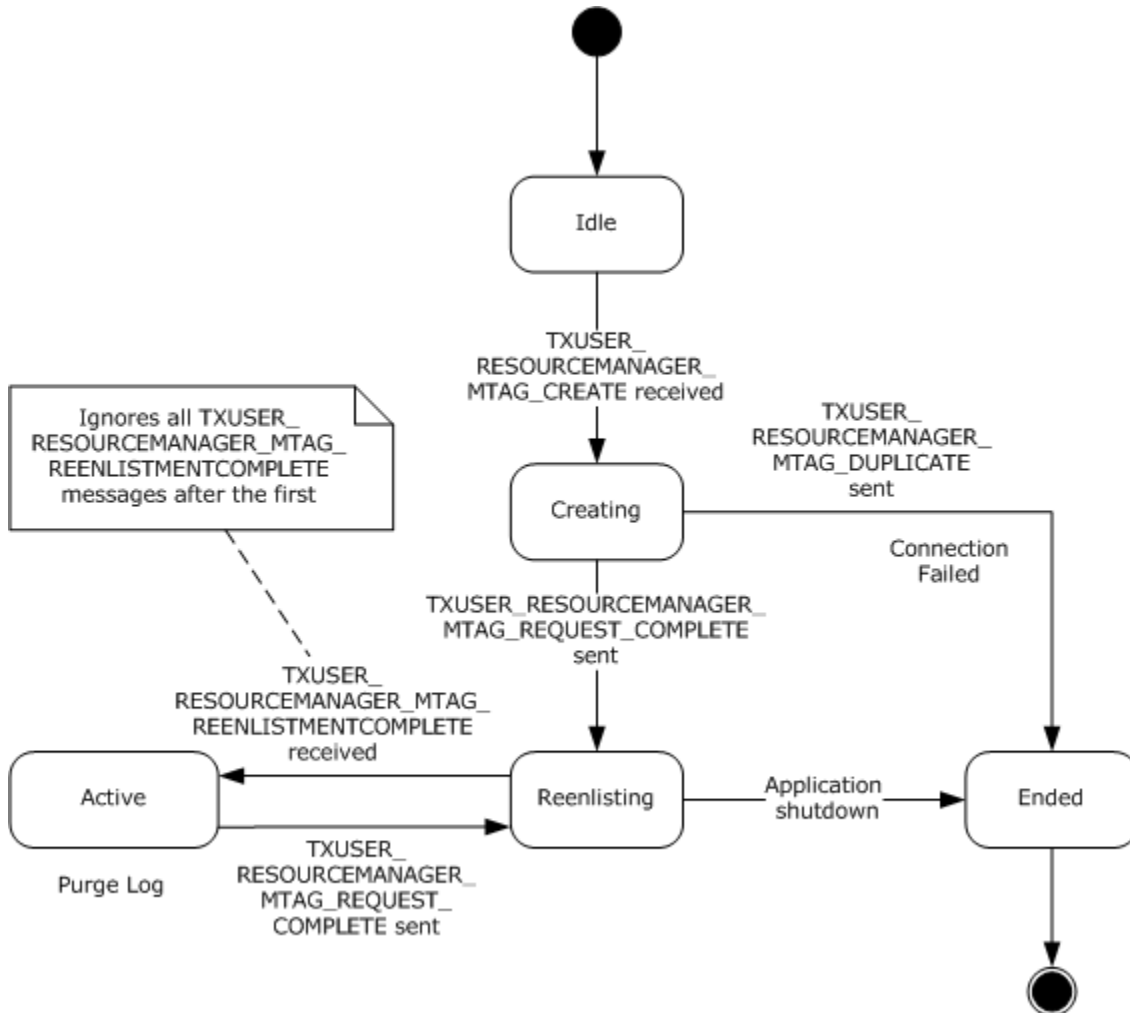


Figure 48: CONNTYPE_TXUSER_RESOURCEMANAGER acceptor states

3.6.1.2 CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL Acceptor States

The transaction manager communicating with a resource manager MUST act as an acceptor for the [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL \(section 2.2.10.1.2\)](#) connection type. In this role, the transaction manager communicating with a resource manager MUST provide support for the following states:

- [Idle \(section 3.6.1.2.1\)](#)
- [Creating \(section 3.6.1.2.2\)](#)
- [Reenlisting \(section 3.6.1.2.3\)](#)
- [Active \(section 3.6.1.2.4\)](#)
- [Ended \(section 3.6.1.2.5\)](#)

3.6.1.2.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_RESOURCEMANAGER_MTAG_CREATE Message \(section 3.6.5.1.1.1\).](#)

3.6.1.2.2 Creating

The following events are processed in this state:

- [Create Resource Manager \(section 3.6.7.9\).](#)

3.6.1.2.3 Reenlisting

The following events are processed in this state:

- [Receiving a TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE Message \(section 3.6.5.1.1.2\).](#)

3.6.1.2.4 Active

The following events are processed in this state:

- [Reenlist Complete \(section 3.6.7.15\).](#)
- [Create Resource Manager \(section 3.6.7.9\).](#)

3.6.1.2.5 Ended

This is the final state.

3.6.1.2.6 State Diagram

The following figure shows the relationship between the [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL \(section 2.2.10.1.2\)](#) acceptor states.

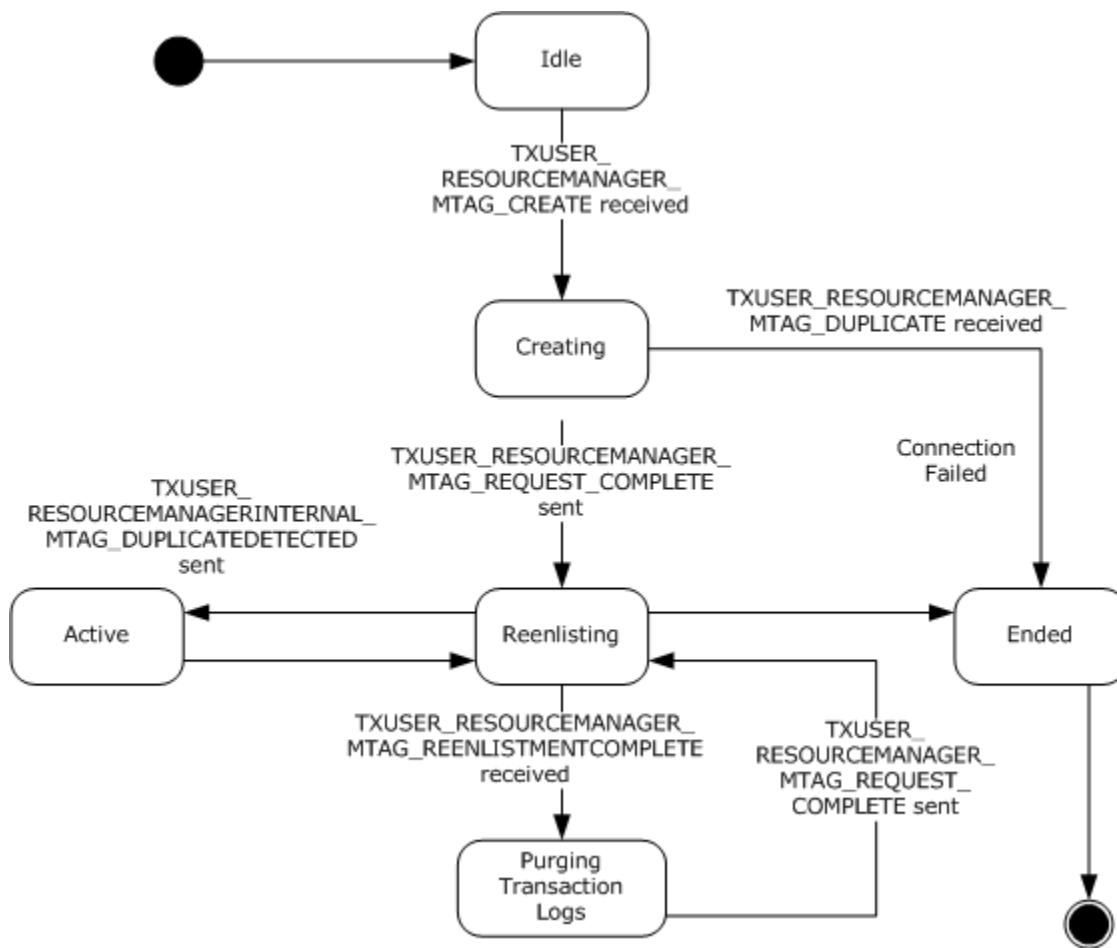


Figure 49: CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL acceptor states

3.6.1.3 CONNTYPE_TXUSER_PHASE0 Acceptor States

The transaction manager communicating with a resource manager MUST act as an acceptor for the [CONNTYPE_TXUSER_PHASE0 \(section 2.2.10.2.1\)](#) connection type. In this role, the transaction manager communicating with a resource manager MUST provide support for the following states:

- [Idle \(section 3.6.1.3.1\)](#)
- [Processing Enlistment Request \(section 3.6.1.3.2\)](#)
- [Active \(section 3.6.1.3.3\)](#)
- [Awaiting Phase Zero Response \(section 3.6.1.3.4\)](#)
- [Ended \(section 3.6.1.3.5\)](#)

3.6.1.3.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_PHASE0_MTAG_CREATE Message \(section 3.6.5.2.1.1\).](#)

3.6.1.3.2 Processing Enlistment Request

The following events are processed in this state:

- [Create Phase Zero Enlistment Success \(section 3.6.7.8\)](#).
- [Create Phase Zero Enlistment Failure \(section 3.6.7.7\)](#).

3.6.1.3.3 Active

The following events are processed in this state:

- [Begin Phase Zero \(section 3.6.7.4\)](#).
- [Phase Zero Aborted \(section 3.6.7.14\)](#).
- [Receiving a TXUSER PHASE0 MTAG UNENLIST Message \(section 3.6.5.2.1.3\)](#).

3.6.1.3.4 Awaiting Phase Zero Response

The following events are processed in this state:

- [Receiving a TXUSER PHASE0 MTAG PHASE0REQDONE Message \(section 3.6.5.2.1.2\)](#).
- [Receiving a TXUSER PHASE0 MTAG UNENLIST Message \(section 3.6.5.2.1.3\)](#).

3.6.1.3.5 Ended

This is the final state.

3.6.1.3.6 State Diagram

The following figure shows the relationship between the [CONNTYPE TXUSER PHASE0 \(section 2.2.10.2.1\)](#) acceptor states.

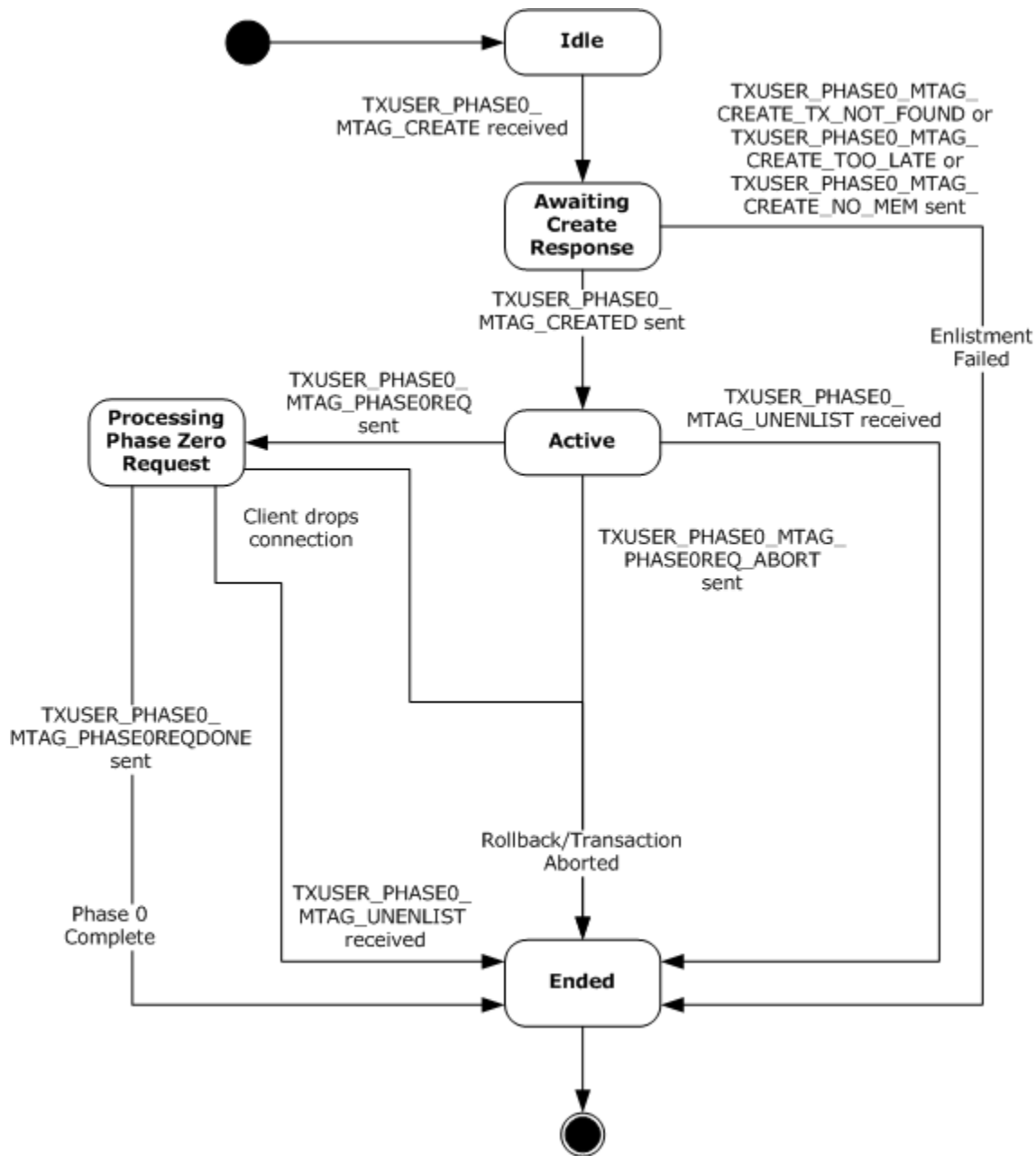


Figure 50: CONNTYPE_TXUSER_PHASE0 acceptor states

3.6.1.4 CONNTYPE_TXUSER_ENLISTMENT Acceptor States

The transaction manager communicating with a resource manager MUST act as an acceptor for the [CONNTYPE_TXUSER_ENLISTMENT \(section 2.2.10.2.2\)](#) connection type. In this role, the transaction manager communicating with a resource manager MUST provide support for the following states:

- [Idle \(section 3.6.1.4.1\)](#)

- [Processing Enlistment Request \(section 3.6.1.4.2\)](#)
- [Active \(section 3.6.1.4.3\)](#)
- [Awaiting Single Phase Commit Response \(section 3.6.1.4.4\)](#)
- [Awaiting Prepare Response \(section 3.6.1.4.5\)](#)
- [Awaiting Prepare Response Aborted \(section 3.6.1.4.6\)](#)
- [Prepared \(section 3.6.1.4.7\)](#)
- [Awaiting Commit Response \(section 3.6.1.4.8\)](#)
- [Awaiting Abort Response \(section 3.6.1.4.9\)](#)
- [Ended \(section 3.6.1.4.10\)](#)

3.6.1.4.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_ENLISTMENT_MTAG_ENLIST Message \(section 3.6.5.2.2.1\).](#)

3.6.1.4.2 Processing Enlistment Request

The following events are processed in this state:

- [Create Subordinate Enlistment Success \(section 3.6.7.11\).](#)
- [Create Subordinate Enlistment Failure \(section 3.6.7.10\).](#)

3.6.1.4.3 Active

The following events are processed in this state:

- [Begin Phase One \(section 3.6.7.3\).](#)
- [Begin Rollback \(section 3.6.7.5\).](#)

3.6.1.4.4 Awaiting Single-Phase Commit Response

The following events are processed in this state:

- [Receiving a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE Message \(section 3.6.5.2.2.2\).](#)

3.6.1.4.5 Awaiting Prepare Response

The following events are processed in this state:

- [Receiving a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE Message \(section 3.6.5.2.2.2\).](#)
- [Begin Rollback \(section 3.4.7.5\).](#)

3.6.1.4.6 Awaiting Prepare Response Aborted

The following events are processed in this state:

- [Receiving a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE Message \(section 3.6.5.2.2.2\).](#)

3.6.1.4.7 Prepared

The following events are processed in this state:

- [Begin Commit \(section 3.6.7.1\).](#)
- [Begin Rollback \(section 3.6.7.5\).](#)

3.6.1.4.8 Awaiting Commit Response

The following events are processed in this state:

- [Receiving a TXUSER_ENLISTMENT_MTAG_COMMITREQDONE Message \(section 3.6.5.2.2.3\).](#)

3.6.1.4.9 Awaiting Abort Response

The following events are processed in this state:

- [Receiving a TXUSER_ENLISTMENT_MTAG_ABORTREQDONE Message \(section 3.6.5.2.2.4\).](#)

3.6.1.4.10 Ended

This is the final state.

3.6.1.4.11 State Diagram

The following figure shows the relationship between the [CONNTYPE_TXUSER_ENLISTMENT \(section 2.2.10.2.2\)](#) acceptor states.

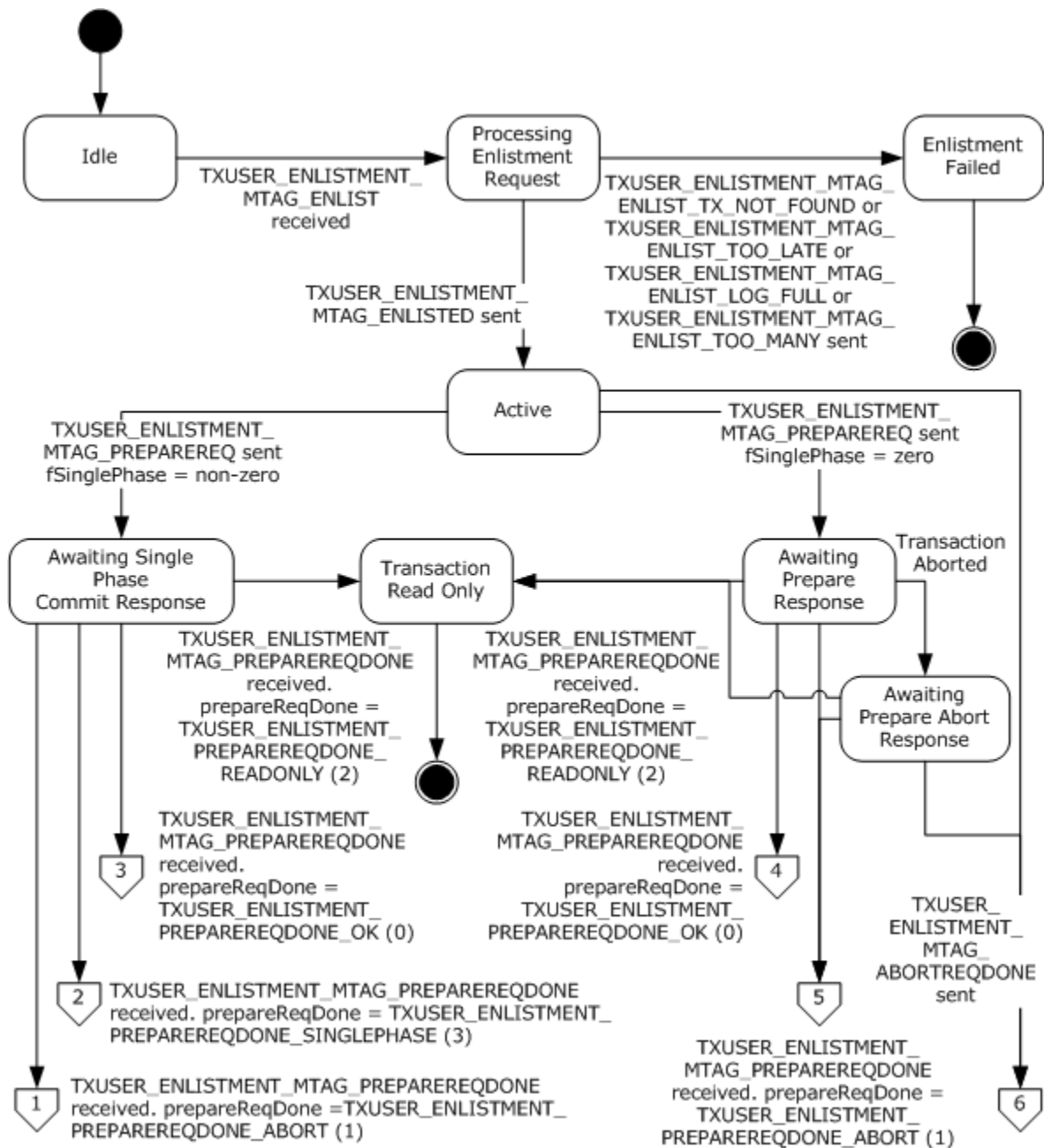


Figure 51: CONNTYPE_TXUSER_ENLISTMENT acceptor states (processing enlistment request)

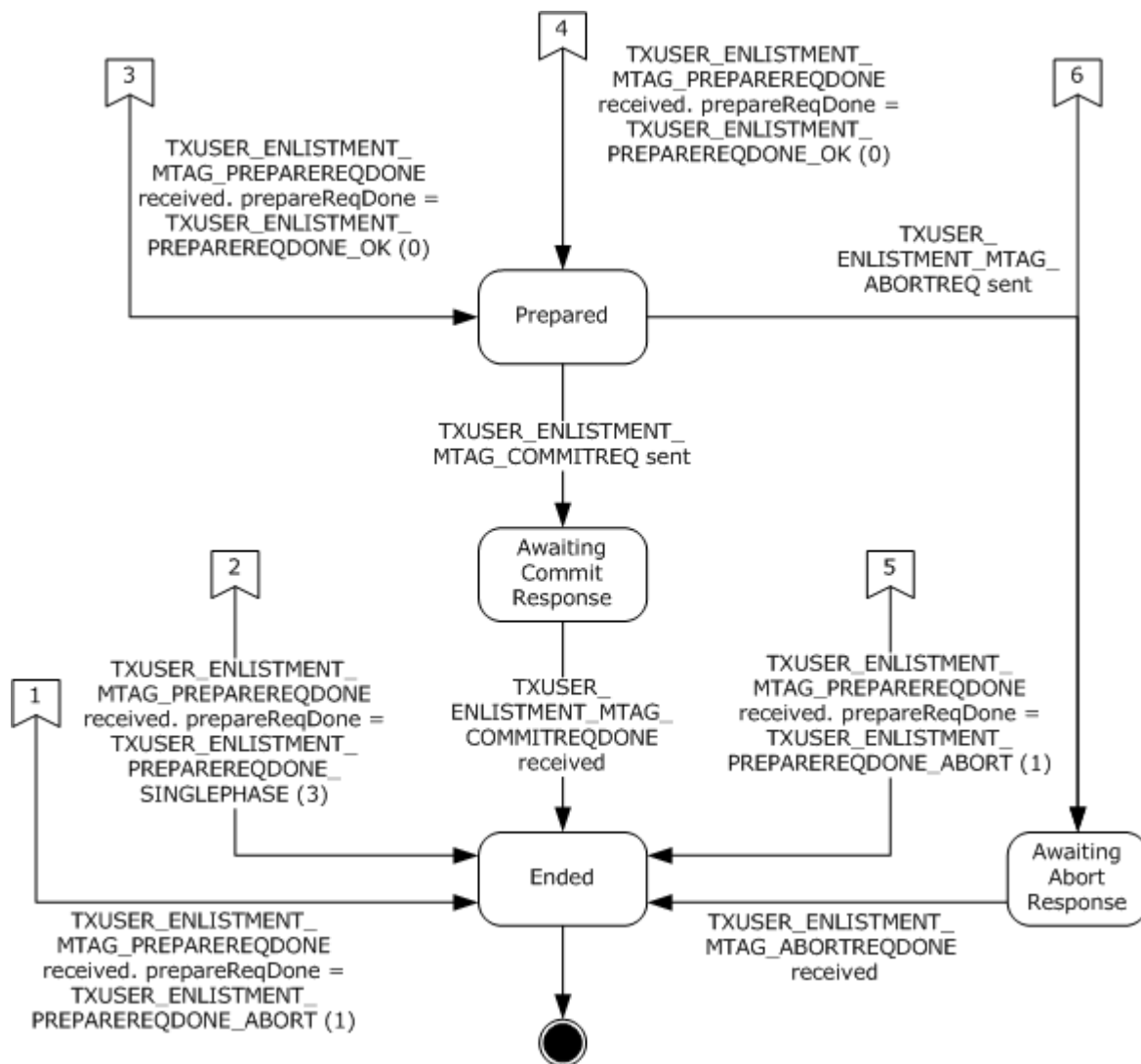


Figure 52: CONNTYPE_TXUSER_ENLISTMENT acceptor states (active)

3.6.1.5 CONNTYPE_TXUSER_REENLIST Acceptor States

The transaction manager communicating with a resource manager MUST act as an acceptor for the [CONNTYPE_TXUSER_REENLIST \(section 2.2.10.3.1\)](#) connection type. In this role, the transaction manager communicating with a resource manager MUST provide support for the following states:

- [Idle \(section 3.6.1.5.1\)](#)
- [Processing Reenlist Request \(section 3.6.1.5.2\)](#)
- [Ended \(section 3.6.1.5.3\)](#)

3.6.1.5.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER_ENLISTMENT_MTAG_ENLIST Message \(section 3.6.5.2.2.1\).](#)

3.6.1.5.2 Processing Reenlist Request

The following events are processed in this state:

- [Begin Commit \(section 3.7.7.1\).](#)
- [Begin Rollback \(section 3.7.7.4\).](#)
- [Reenlist Time-Out Timer \(section 3.6.2.1\).](#)

3.6.1.5.3 Ended

This is the final state.

3.6.1.5.4 State Diagram

The following figure shows the relationship between the [CONNTYPE_TXUSER_REENLIST \(section 2.2.10.3.1\)](#) acceptor states.

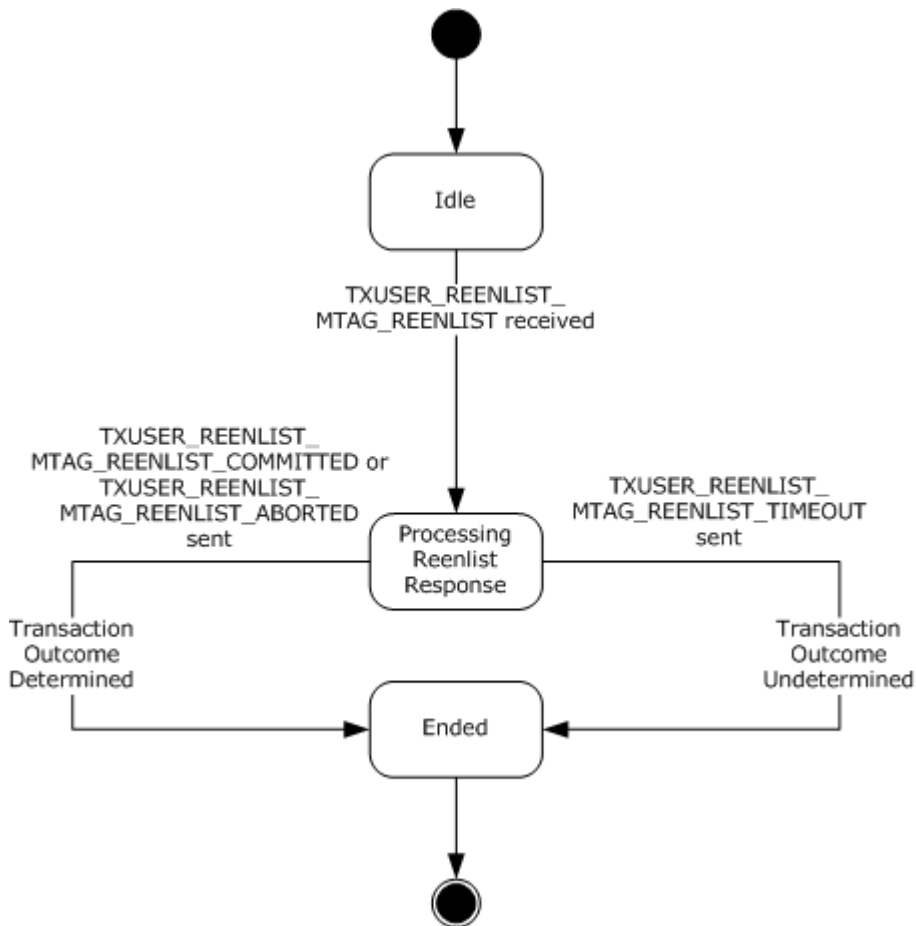


Figure 53: CONNTYPE_TXUSER_REENLIST acceptor states

3.6.1.6 CONNTYPE_TXUSER_VOTER Acceptor States

The transaction manager communicating with a resource manager MUST act as an acceptor for the [CONNTYPE TXUSER VOTER \(section 2.2.10.4.1\)](#) connection type. In this role, the transaction manager communicating with a resource manager MUST provide support for the following states:

- [Idle \(section 3.6.1.6.1\)](#)
- [Create Voter \(section 3.6.1.6.2\)](#)
- [Active \(section 3.6.1.6.3\)](#)
- [Awaiting Voter Response \(section 3.6.1.6.4\)](#)
- [Awaiting Outcome \(section 3.6.1.6.5\)](#)
- [Ended \(section 3.6.1.6.6\)](#)

3.6.1.6.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a TXUSER VOTER MTAG CREATE Message \(section 3.6.5.4.1.1\).](#)

3.6.1.6.2 Create Voter

The following events are processed in this state:

- [Create Voter Enlistment Success \(section 3.6.7.13\).](#)
- [Create Voter Enlistment Failure \(section 3.6.7.12\).](#)

3.6.1.6.3 Active

The following events are processed in this state:

- [Begin Voting \(section 3.6.7.6\).](#)
- [Begin Rollback \(section 3.6.7.5\).](#)

3.6.1.6.4 Awaiting Voter Response

The following events are processed in this state:

- [Receiving a TXUSER VOTER MTAG VOTERREQDONE Message \(section 3.6.5.4.1.2\).](#)

3.6.1.6.5 Awaiting Outcome

The following events are processed in this state:

- [Begin Commit \(section 3.6.7.1\).](#)
- [Begin Rollback \(section 3.6.7.5\).](#)
- [Begin In Doubt \(section 3.6.7.2\).](#)

3.6.1.6.6 Ended

This is the final state.

3.6.1.6.7 State Diagram

The following figure shows the relationship between the [CONNTYPE_TXUSER_VOTER](#) (section 2.2.10.4.1) acceptor states.

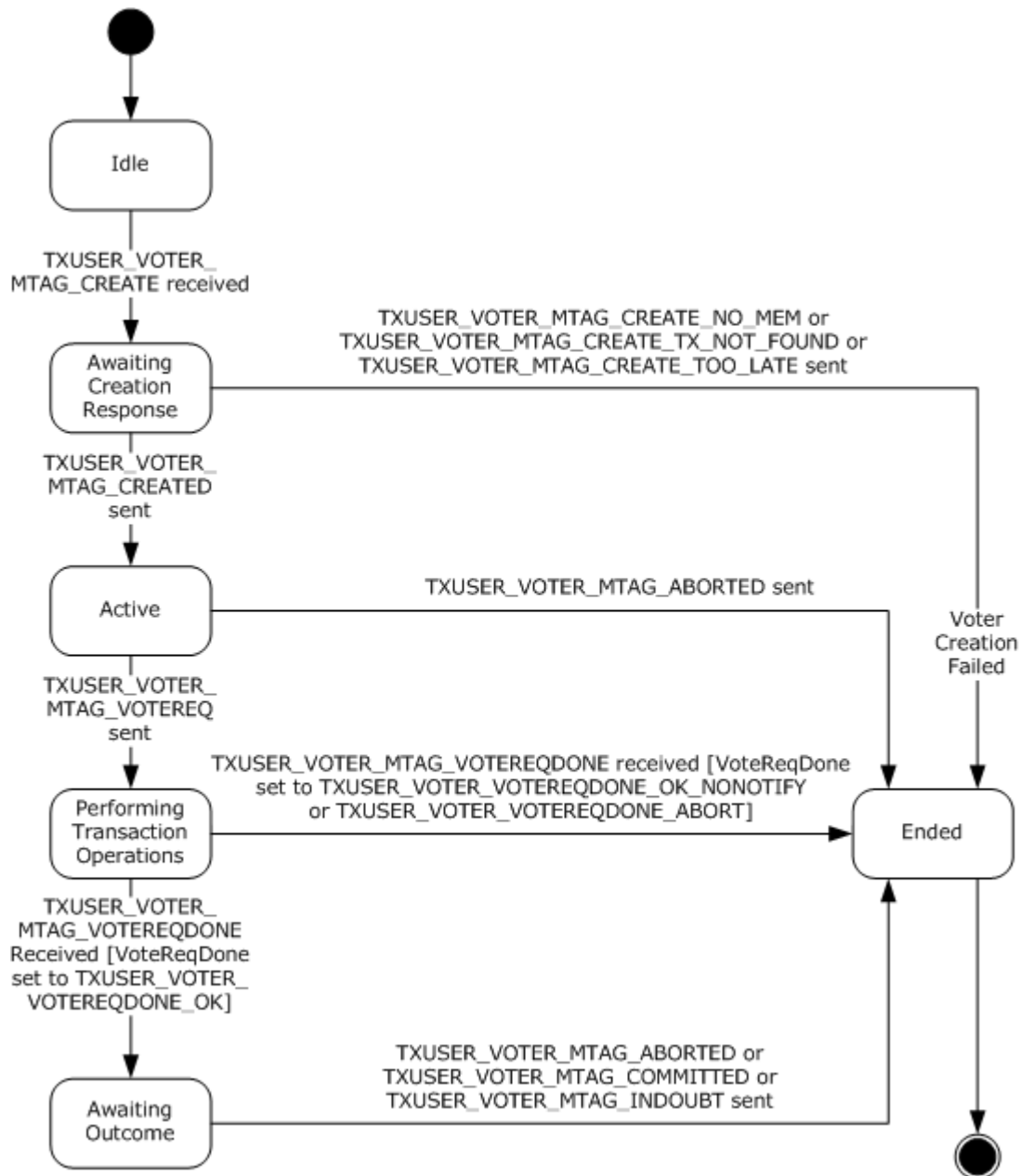


Figure 54: CONNTYPE_TXUSER_VOTER acceptor states

3.6.2 Timers

The transaction manager communicating with a resource manager facet **MUST** provide the timer that is shown in the next section.

3.6.2.1 Reenlist Time-Out Timer

The timer **MUST** be set when the transaction manager communicating with a resource manager facet receives a [TXUSER_REENLIST_MTAG_REENLIST \(section 2.2.10.3.1.1\)](#) message on a [CONNTYPE_TXUSER_REENLIST \(section 2.2.10.3.1\)](#) connection. The timer **MUST** be canceled when the CONNTYPE_TXUSER_REENLIST connection is disconnected.

The timer has no default value. The initial value of the timer **MUST** be provided in the TXUSER_REENLIST_MTAG_REENLIST message. The minimum value of the timer **MUST** be zero, which means that the timer never generates a timer event. In this case, the [Reenlist Time-Out Timer Event \(section 3.6.6.1\)](#) is never signaled, and the timeout reply message triggered by this event is never sent.

When the timer is initialized, the transaction manager communicating with a resource manager facet **MUST** provide an Enlistment object to associate with the timer. When the timer expires, the same Enlistment object **MUST** be provided with the timer notification. The transaction manager communicating with a resource manager facet **MUST** provide a distinct Reenlist Timeout timer instance for each CONNTYPE_TXUSER_REENLIST connection.

3.6.3 Initialization

When the transaction manager communicating with a resource manager facet is initialized:

- The transaction manager communicating with a resource managerfacet **MUST** examine the following security flags on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) and perform the following actions:
 - If either the Allow Network Access flag or the Allow Remote Clients flag is set to false:
 - The transaction manager communicating with a resource managerfacet **MUST** refuse to accept incoming connections from remote machines for the following connection types:
 - [CONNTYPE_TXUSER_ENLISTMENT \(section 2.2.10.2.2\)](#)
 - [CONNTYPE_TXUSER_RESOURCEMANAGER \(section 2.2.10.1.1\)](#)
 - [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL \(section 2.2.10.1.2\)](#)
 - [CONNTYPE_TXUSER_REENLIST \(section 2.2.10.3.1\)](#)
 - [CONNTYPE_TXUSER_VOTER \(section 2.2.10.4.1\)](#)
 - [CONNTYPE_TXUSER_PHASE0 \(section 2.2.10.2.1\)](#)

3.6.4 Higher-Layer Triggered Events

None.

3.6.5 Message Processing Events and Sequencing Rules

3.6.5.1 Resource Manager Registration

3.6.5.1.1 CONNTYPE_TXUSER_RESOURCEMANAGER as Acceptor

For all messages that are received in this connection type, the transaction manager communicating with a resource manager facet **MUST** process the message as specified in section [3.1](#). The transaction manager communicating with a resource manager facet **MUST** additionally follow the processing rules as specified in the following sections.

3.6.5.1.1.1 Receiving a TXUSER_RESOURCEMANAGER_MTAG_CREATE Message

When the transaction manager communicating with a resource manager facet receives a [TXUSER_RESOURCEMANAGER_MTAG_CREATE \(section 2.2.10.1.1.1\)](#) message, the transaction manager communicating with a resource manager facet **MUST** perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Creating.
 - Create a resource manager object using the following values:
 - The **guidRM** field from the message as the resource manager identifier.
 - The **guidSession** field from the message as the session identifier of the resource manager.
 - The current connection.
 - Assign the resource manager object to the connection-specific data field of the connection.
 - Signal the [Create Resource Manager \(section 3.6.7.9\)](#) event on the transaction manager communicating with a resource manager facet with the resource manager object.
- Otherwise, the message **MUST** be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.1.1.2 Receiving a TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE Message

When the transaction manager communicating with a resource manager facet receives a [TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE \(section 2.2.10.1.1.3\)](#) message, the transaction manager communicating with a resource manager **MUST** perform the following actions:

- If the connection state is Reenlisting:
 - Set the connection state to Active.
 - Retrieve the resource manager object that is referenced by the connection.
 - Signal the [Reenlist Complete \(section 3.6.7.15\)](#) event on the transaction manager communicating with a resource manager facet with the resource manager object that is referenced by the Connection-Specific Data field of the connection.
- Otherwise, the message **MUST** be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.1.1.3 Connection Disconnected

When a [CONNTYPE_TXUSER_RESOURCEMANAGER \(section 2.2.10.1.1\)](#) connection is disconnected, the transaction manager communicating with a resource manager facet MUST:

- Set the connection state to Ended.
- Signal the [Resource Manager Down \(section 3.6.7.9\)](#) event on the transaction manager communicating with a resource manager facet with the resource manager object referenced by the Connection-Specific Data field of the connection.

3.6.5.1.2 CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL as Acceptor

For all messages received in this connection type, the transaction manager communicating with a resource manager facet MUST process the message as specified in section [3.1](#). The transaction manager communicating with a resource manager facet MUST additionally follow the processing rules as specified in the following sections.

3.6.5.1.2.1 Receiving a TXUSER_RESOURCEMANAGER_MTAG_CREATE Message

When the transaction manager communicating with a resource manager facet receives a [TXUSER_RESOURCEMANAGER_MTAG_CREATE](#) message, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Creating.
 - Create a resource manager object using the following values:
 - The **guidRM** field from the message as the resource manager identifier.
 - The **guidSession** field from the message as the session identifier of the resource manager.
 - This connection.
 - Assign the resource manager object to the connection-specific data field of the connection.
 - Signal the [Create Resource Manager \(section 3.6.7.9\)](#) event on the transaction manager communicating with a resource manager facet with the resource manager object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.1.2.2 Receiving a TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE Message

When the transaction manager communicating with a resource manager facet receives a [TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE \(section 2.2.10.1.1.3\)](#) message, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- If the connection state is Reenlisting:
 - Set the connection state to Active.

- Signal the [Reenlist Complete \(section 3.6.7.15\)](#) event on the transaction manager communicating with a resource manager facet with the resource manager object that is referenced by the connection-specific data field of the connection.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.1.2.3 Connection Disconnected

When a [CONNTYPE_TXUSER_RESOURCEMANAGER \(section 2.2.10.1.1\)](#) connection is disconnected, the transaction manager communicating with a resource manager facet MUST:

- Set the connection state to Ended.
- Signal the [Resource Manager Down \(section 3.6.7.16\)](#) event on the transaction manager communicating with a resource manager facet with the resource manager object referenced by the connection-specific data field of the connection.

3.6.5.2 Transaction Coordination

3.6.5.2.1 CONNTYPE_TXUSER_PHASE0 as Acceptor

For all messages received in this connection type, the transaction manager communicating with a resource manager facet MUST process the message as specified in section [3.1](#). The transaction manager communicating with a resource manager facet MUST additionally follow the processing rules as specified in the following sections.

3.6.5.2.1.1 Receiving a TXUSER_PHASE0_MTAG_CREATE Message

When the transaction manager receives a [TXUSER_PHASE0_MTAG_CREATE](#) message, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Enlistment Response.
 - Find the transaction object in the transaction table of the transaction manager using the **guidTx** field from the message.
 - If the transaction is not found:
 - Send a [TXUSER_PHASE0_MTAG_CREATE_TX_NOT_FOUND](#) message using the connection.
 - Set the connection state to [Ended](#).
- Otherwise:
 - Create a new Enlistment object with the following values:
 - The transaction manager communicating with a resource manager facet.
 - The transaction object.
 - The connection.
 - Assign the new Enlistment object to the **enlistment** field of the connection.

- Signal the [Create Phase Zero Enlistment](#) event on the [Core Transaction Manager Facet](#) with the Enlistment object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.2.1.2 Receiving a TXUSER_PHASE0_MTAG_PHASE0REQDONE Message

When the transaction manager receives a [TXUSER_PHASE0_MTAG_PHASE0REQDONE](#) message, the transaction manager MUST perform the following actions:

- If the connection state is Awaiting Phase Zero Response:
 - Signal the [Enlistment Phase Zero Complete](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The Enlistment object that is referenced by this connection.
 - The completed outcome value.
 - Set the connection state to [Ended](#).
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.2.1.3 Receiving a TXUSER_PHASE0_MTAG_UNENLIST Message

When the transaction manager receives a [TXUSER_PHASE0_MTAG_UNENLIST](#) (section [2.2.10.2.1.8](#)) message, the transaction manager MUST perform the following actions:

- If the connection state is Active:
 - Signal the [Unenlist Phase Zero Enlistment](#) (section [3.2.7.34](#)) event on the [Core Transaction Manager Facet](#) (section [1.3.3.3.1](#)) with the Enlistment object that is referenced by this connection.
 - Set the connection state to [Ended](#) (section [3.2.1.4.14](#)).
- If the connection state is Awaiting Phase Zero Response:
 - Signal the [Enlistment Phase Zero Complete](#) (section [3.2.7.17](#)) event on the Core Transaction Manager Facet with the following arguments:
 - The Enlistment object that is referenced by this connection.
 - The completed outcome value.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.2.1.4 Connection Disconnected

When a [CONNTYPE_TXUSER_PHASE0](#) (section [2.2.10.2.1](#)) connection is disconnected, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- If the connection state is Awaiting Phase Zero Response:

- Signal the [Enlistment Phase Zero Complete \(section 3.2.7.17\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the following arguments:
 - The Enlistment object that is referenced by this connection.
 - The aborted outcome value.
- Otherwise, if the connection state is [Active \(section 3.2.1.4.2\)](#):
 - Signal the [Enlistment Unilaterally Aborted \(section 3.2.7.19\)](#) event on the Core Transaction Manager Facet with the Enlistment object that is referenced by this connection.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.6.5.2.2 CONNTYPE_TXUSER_ENLISTMENT as Acceptor

For all messages that are received in this connection type, the transaction manager MUST process the message as specified in section [3.1](#). The transaction manager MUST additionally follow the processing rules as specified in the following sections.

3.6.5.2.2.1 Receiving a TXUSER_ENLISTMENT_MTAG_ENLIST Message

When the transaction manager receives a [TXUSER_ENLISTMENT_MTAG_ENLIST](#) message, the resource manager MUST perform the following actions:

- If the connection state is Idle:
 - Find the transaction object in the transaction table of the transaction manager using the **guidTx** field from the message.
 - If the transaction is not found:
 - Send a [TXUSER_ENLIST_MTAG_ENLIST_TX_NOT_FOUND](#) message using the connection.
 - Set the connection state to Ended.
 - Otherwise:
 - Find the resource manager object in the transaction manager's active resource manager table using the **guidRm** field from the message.
 - If the resource manager is not found:
 - Send a [TXUSER_ENLIST_MTAG_ENLIST_TOO_LATE](#) message using the connection.
 - Set the connection state to Ended.
 - Otherwise:
 - Set the connection state to Processing Enlistment Request.
 - Create a new enlistment object with the following values:
 - The transaction manager communicating with a resource manager facet.
 - The transaction object.
 - The connection.

- The identifier field of the resource manager object.
- Signal the [Create Subordinate Enlistment](#) event on the [Core Transaction Manager Facet](#) with the new enlistment object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.2.2.2 Receiving a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE Message

When the transaction manager receives a [TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE](#) (section [2.2.10.2.2.12](#)) message, the transaction manager MUST perform the following actions:

- If the connection state is Awaiting Prepare Response Aborted:
 - If the **prepareReqDone** field of the message is TXUSER_ENLISTMENT_PREPAREREQDONE_OK:
 - Send a [TXUSER_ENLISTMENT_MTAG_ABORTREQ](#) (section [2.2.10.2.2.1](#)) message using the connection.
 - Set the connection state to Awaiting Abort Response
 - Otherwise, set the connection state to Ended.
- If the connection state is Awaiting Single Phase Commit:
 - Signal the [Enlistment Phase One Complete](#) (section [3.2.7.16](#)) event on the [Core Transaction Manager Facet](#) (section [1.3.3.3.1](#)) with the following arguments:
 - The Enlistment object of the connection.
 - The [Phase One](#) outcome set to:
 - Committed if the **prepareReqDone** field from the message is TXUSER_ENLISTMENT_PREPAREREQDONE_SINGLEPHASE.
 - Aborted if the **prepareReqDone** field from the message is TXUSER_ENLISTMENT_PREPAREREQDONE_ABORT.
 - Read-only if the **prepareReqDone** field from the message is TXUSER_ENLISTMENT_PREPAREREQDONE_READ_ONLY.
 - Prepared if the **prepareReqDone** field from the message is TXUSER_ENLISTMENT_PREPAREREQDONE_OK.
 - If the **prepareReqDone** field from the message is set to TXUSER_ENLISTMENT_PREPAREREQDONE_OK:
 - Set the connection state to Prepared.
 - Otherwise:
 - Set the connection state to Ended.
- If the connection state is Awaiting Prepare Response:

- Signal the Enlistment Phase One Complete (section 3.2.7.16) event on the Core Transaction Manager Facet with the following arguments:
 - The Enlistment object of the connection.
 - The Phase One outcome set to:
 - Aborted if the **prepareReqDone** field is TXUSER_ENLISTMENT_PREPAREREQDONE_ABORT.
 - Read-only if the **prepareReqDone** field is TXUSER_ENLISTMENT_PREPAREREQDONE_READ_ONLY.
 - Prepared if the **prepareReqDone** field is TXUSER_ENLISTMENT_PREPAREREQDONE_OK.
 - If the **prepareReqDone** field from the message is set to TXUSER_ENLISTMENT_PREPAREREQDONE_OK:
 - Set the connection state to Prepared.
 - Otherwise:
 - Set the connection state to Ended.
- If the connection state is Awaiting Abort Response:
 - Ignore the message.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.2.2.3 Receiving a TXUSER_ENLISTMENT_MTAG_COMMITREQDONE Message

When the transaction manager receives a [TXUSER_ENLISTMENT_MTAG_COMMITREQDONE](#) (section [2.2.10.2.2.4](#)) message, the transaction manager MUST perform the following action:

- If the connection state is Awaiting Commit Response:
 - Signal the [Enlistment Commit Complete](#) (section [3.2.7.15](#)) event on the [Core Transaction Manager Facet](#) (section [1.3.3.3.1](#)) with the enlistment object of the connection.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.2.2.4 Receiving a TXUSER_ENLISTMENT_MTAG_ABORTREQDONE Message

When the transaction manager receives a [TXUSER_ENLISTMENT_MTAG_ABORTREQDONE](#) message, the transaction manager MUST perform the following action:

- If the connection state is Awaiting Abort Response:
 - Signal the [Enlistment Rollback Complete](#) event on the [Core Transaction Manager Facet](#) with the enlistment object of the connection.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.2.2.5 Connection Disconnected

When a [CONNTYPE_TXUSER_ENLISTMENT \(section 2.2.10.2.2\)](#) connection is disconnected, the transaction manager MUST perform the following actions:

- If the connection state is either Processing Enlistment Request or [Active \(section 3.2.1.4.2\)](#):
 - Signal the [Enlistment Unilaterally Aborted \(section 3.2.7.19\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the Enlistment object of the connection.
- Otherwise, if the connection state is Awaiting Prepare Response:
 - Signal the [Phase One Complete \(section 3.4.7.13\)](#) event on the Core Transaction Manager Facet with the following arguments:
 - The Enlistment object of the connection.
 - The aborted outcome.
- Otherwise, if the connection state is Awaiting Single Phase Commit Response:
 - Signal the [Enlistment Single-Phase Commit Request Completed \(section 3.5.4.7\)](#) event on the Core Transaction Manager Facet with the following arguments:
 - The Enlistment object of the connection.
 - The [In Doubt \(section 3.2.1.4.12\)](#) outcome.
- Otherwise, if the connection state is Awaiting Commit Response, the transaction manager MUST perform the following action:
 - Add the Enlistment object of the transaction manager to the Failed to Notify list of the transaction manager.
- Otherwise, if the connection state is Awaiting Abort Response:
 - Signal the [Enlistment Rollback Complete \(section 3.2.7.18\)](#) event on the Core Transaction Manager Facet with the enlistment object of the connection.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.6.5.3 Transaction Recovery

3.6.5.3.1 CONNTYPE_TXUSER_REENLIST as Acceptor

For all messages received in this connection type, the transaction manager communicating with a resource manager facet MUST process the message as specified in section [3.1](#). The transaction manager communicating with a resource manager facet MUST additionally follow the processing rules as specified in the following sections.

3.6.5.3.1.1 Receiving a TXUSER_REENLIST_MTAG_REENLIST Message

When the transaction manager communicating with a resource manager facet receives a [TXUSER_REENLIST_MTAG_REENLIST](#) message, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- If the connection state is Idle:

- Set the connection state to Processing Request Response.
- Look up a resource manager object in the active resource manager table, using the **guidRm** field from the message as the key.
- If the resource manager does not exist:
 - Send a [TXUSER_REENLIST_MTAG_REENLIST_ABORTED](#) message using the connection.
 - Set the connection state to [Ended](#).
- Otherwise:
 - Look up a transaction object in the transaction table using the **guidTx** field from the message as the key.
 - If the transaction is not found:
 - Send a TXUSER_REENLIST_MTAG_REENLIST_ABORTED message using the connection.
 - Set the connection state to Ended.
 - Find an Enlistment object in the transaction object's Phase Two Enlistment list whose resource manager field matches the resource manager object.
 - If no Enlistment object is found:
 - Send a TXUSER_REENLIST_MTAG_REENLIST_ABORTED message using the connection.
 - Set the connection state to Ended.
 - Otherwise:
 - Initialize the reenlist time-out timer providing the following arguments:
 - The **ulTimeout** field from the message.
 - The Enlistment object that is found in the Phase Two enlistment list.
 - Assign the Enlistment object to the **enlistment** field of the connection.
 - Assign the connection object to the connection field of the enlistment.
 - Signal the [Request Transaction Outcome](#) event on the [Core Transaction Manager Facet](#) with the new Enlistment object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.3.1.2 Connection Disconnected

This event MUST be processed as specified in section [3.1.8.1](#).

3.6.5.4 Voting

3.6.5.4.1 CONNTYPE_TXUSER_VOTER as Acceptor

For all messages that are received in this connection type, the transaction manager that is communicating with a resource manager facet MUST process the message as specified in section

[3.1](#). The transaction manager communicating with a resource manager facet MUST additionally follow the processing rules as specified in the following sections.

3.6.5.4.1.1 Receiving a TXUSER_VOTER_MTAG_CREATE Message

When the transaction manager communicating with a resource manager facet receives a [TXUSER_VOTER_MTAG_CREATE](#) message, it MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Creating Voter.
 - Find the transaction object in the transaction manager's transaction table using the **guidTx** field from the message.
 - If the transaction is not found:
 - Send a [TXUSER_VOTER_MTAG_CREATE_TX_NOT_FOUND](#) message using the connection.
 - Set the connection state to [Ended](#).
- Otherwise:
 - Create a new Enlistment object with the following values:
 - The transaction manager communicating with a resource manager facet.
 - The transaction object.
 - The connection.
 - Assign the new enlistment object to the **enlistment** field of the connection.
 - Signal the [Create Voter Enlistment](#) event on the [Core Transaction Manager Facet](#) with the Enlistment object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.4.1.2 Receiving a TXUSER_VOTER_MTAG_VOTERREQDONE Message

When the transaction manager communicating with a resource manager facet receives a [TXUSER_VOTER_MTAG_VOTERREQDONE](#) message, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- If the connection state is Awaiting Voter Response:
 - Set the connection state as follows:
 - If the **VoteReqDone** field from the message is TXUSER_VOTER_VOTERREQDONE_ABORT or TXUSER_VOTER_VOTERREQDONE_OK_NONOTIFY:
 - Set the connection state to [Ended](#).
 - Otherwise:
 - Set the **VoteReqDone** field from the message to TXUSER_VOTER_VOTERREQDONE_OK.
 - Set the connection state to Awaiting Outcome.

- Signal the [Enlistment Vote Complete](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The Enlistment object that is referenced by this connection.
 - The Vote outcome, which MUST be set to:
 - Prepared if the **VoteReqDone** field from the message is TXUSER_VOTER_VOTEREQDONE_OK.
 - Aborted if the **VoteReqDone** field from the message is TXUSER_VOTER_VOTEREQDONE_ABORT.
 - Read-only if the **VoteReqDone** field from the message is TXUSER_VOTER_VOTEREQDONE_OK_NONOTIFY.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.6.5.4.1.3 Connection Disconnected

When a [CONNTYPE_TXUSER_VOTER \(section 2.2.10.4.1\)](#) connection is disconnected, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- If the connection state is either [Active \(section 3.2.1.4.2\)](#) or Awaiting Voter Response:
 - Signal the [Enlistment Unilaterally Aborted \(section 3.2.7.19\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the Enlistment object that is referenced by this connection.
- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.6.6 Timer Events

3.6.6.1 Reenlist Timeout Timer

When this timer expires, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- Send a [TXUSER_REENLIST_MTAG_REENLIST_TIMEOUT \(section 2.2.10.3.1.4\)](#) message using the transaction that is referenced by the provided Enlistment object.
- Set the transaction state to [Ended \(section 3.2.1.4.14\)](#).

3.6.7 Other Local Events

A transaction manager communicating with a resource manager facet MUST be prepared to process the local events that are defined in the following sections.

The transaction manager communicating with a resource manager MUST be prepared to process local events pertaining to [Phase Zero](#) functionality only on versions where the connection type [CONNTYPE_TXUSER_PHASE0](#) is supported. Section [2.2.1.1.3](#) defines protocol version support for this connection type. The following local events are affected:

- [Create Phase Zero Enlistment Success](#)
- [Create Phase Zero Enlistment Failure](#)

- [Begin Phase Zero](#)
- [Phase Zero Aborted](#)

3.6.7.1 Begin Commit

The Begin Commit event MUST be signaled with the following arguments:

- An Enlistment object.

If the Begin Commit event is signaled, the transaction manager MUST perform the following actions:

- If the connection of the provided enlistment is of type [CONNTYPE_TXUSER_VOTER](#):
 - Send a [TXUSER_STATUS_MTAG_COMMITTED](#) message using the connection.
 - Set the connection state to [Ended](#).
- Otherwise, if the connection of the provided enlistment is of type [CONNTYPE_TXUSER_ENLISTMENT](#):
 - If the connection state is Ended:
 - Add the provided Enlistment object to the [Failed to Notify](#) list of the transaction manager.
 - Otherwise:
 - Send a [TXUSER_ENLISTMENT_MTAG_COMMITREQ](#) message using the connection.
 - Set the connection state to Awaiting Commit Response.
- Otherwise, if the connection of the provided enlistment is of type [CONNTYPE_TXUSER_REENLIST](#):
 - If the connection state is Processing Reenlist Request:
 - Send a [TXUSER_REENLIST_MTAG_REENLIST_COMMITTED](#) message using the connection.
 - Set the connection state to Ended.
 - Otherwise:
 - Ignore the signal.

3.6.7.2 Begin In Doubt

The Begin In Doubt event MUST be signaled with the following arguments:

- An Enlistment object.

If the Begin In Doubt event is signaled, the transaction manager MUST perform the following actions:

- Send a [TXUSER_STATUS_MTAG_INDOUBT](#) message using the connection of the provided enlistment.
- Set the connection state to [Ended](#).

3.6.7.3 Begin Phase One

The Begin Phase One event MUST be signaled with the following arguments:

- An Enlistment object.
- A Boolean value indicating whether the transaction manager communicating with a resource managerfacet SHOULD OR MUST NOT attempt to perform a Single Phase Commit.

If the Begin Phase One event is signaled, the transaction manager MUST perform the following actions:

- If the connection state of the enlistment is Active:
 - If the provided Single Phase Commit flag is true:
 - Send a [TXUSER_ENLISTMENT_MTAG_PREPAREREQ \(section 2.2.10.2.2.11\)](#) message using the connection of the provided enlistment.
 - The **fSinglePhase** field MUST be set to a nonzero value.
 - Set the connection state to Awaiting Single Phase Commit Response.
 - Otherwise:
 - Send a TXUSER_ENLISTMENT_MTAG_PREPAREREQ (section 2.2.10.2.2.11) message using the connection of the provided enlistment:
 - The **fSinglePhase** field MUST be set to 0.
 - Set the connection state to Awaiting Prepare Response.
- Otherwise, ignore the event.

3.6.7.4 Begin Phase Zero

The Begin Phase Zero event MUST be signaled with the following arguments:

- An Enlistment object.

If the Begin Phase Zero event is signaled, the transaction manager MUST perform the following actions:

- If the connection state is Active:
 - Send a [TXUSER_PHASE0_MTAG_PHASE0REQ](#) message using the connection of the provided enlistment.
 - Set the connection state to Awaiting Phase Zero Response.
- Otherwise:
 - Ignore the event.

3.6.7.5 Begin Rollback

The Begin Rollback event MUST be signaled with the following arguments:

- An Enlistment object.

If the Begin Rollback event is signaled, the transaction manager MUST perform the following actions:

- If the connection of the provided enlistment is of type [CONNTYPE_TXUSER_VOTER](#):
 - Send a [TXUSER_STATUS_MTAG_ABORTED](#) message using the connection.
 - Set the connection state to [Ended](#).
- Otherwise, if the connection of the provided enlistment is of type [CONNTYPE_TXUSER_ENLISTMENT](#):
 - If the connection state is Idle:
 - Signal the [Enlistment Rollback Complete](#) event on the [Core Transaction Manager Facet](#) with the provided Enlistment object.
 - Otherwise:
 - If the connection state is Active:
 - Send a [TXUSER_ENLISTMENT_MTAG_ABORTREQ](#) message using the connection.
 - Set the connection state to Awaiting Abort Response.
 - Otherwise, if the connection state is Awaiting Prepare Response:
 - Set the connection state to Awaiting Prepare Response Aborted.
- Otherwise, if the connection of the provided enlistment is of type [CONNTYPE_TXUSER_REENLIST](#):
 - If the connection state is Processing Reenlist Request:
 - Send a [TXUSER_REENLIST_MTAG_REENLIST_ABORTED](#) message using the connection.
 - Set the connection state to Ended.
 - Otherwise, ignore the signal.

3.6.7.6 Begin Voting

The Begin Voting event MUST be signaled with the following arguments:

- An Enlistment object.

If the Begin Voting event is signaled, the transaction manager MUST perform the following actions:

- If the connection state is Active:
 - Send a [TXUSER_VOTER_MTAG_VOTERREQ](#) message using the connection of the provided enlistment.
 - Set the connection state to Awaiting Voter Response.
- Otherwise, ignore the event.

3.6.7.7 Create Phase Zero Enlistment Failure

The Create Phase Zero Enlistment Success event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:
 - Too Late
 - Tx Not Found

If the Create Phase Zero Enlistment Success event is signaled, the transaction manager MUST perform the following actions:

- If the connection state is Processing Enlistment Request:
 - Send the matching message for the following reason codes using the connection of the provided enlistment:
 - Too Late: [TXUSER PHASE0 MTAG CREATE TOO LATE \(section 2.2.10.2.1.2\)](#).
 - Tx Not Found: [TXUSER PHASE0 MTAG CREATE TX NOT FOUND \(section 2.2.10.2.1.3\)](#)
 - Set the connection state to [Ended \(section 3.2.1.4.14\)](#).
- Otherwise, ignore the event.

3.6.7.8 Create Phase Zero Enlistment Success

The Create Phase Zero Enlistment Success event MUST be signaled with the following arguments:

- An Enlistment object.

If the Create Phase Zero Enlistment Success event is signaled, the transaction manager MUST perform the following actions:

- If the connection state is Processing Enlistment Request:
 - Send a [TXUSER PHASE0 MTAG CREATED](#) message using the connection of the provided enlistment.
 - Set the connection state to Active.
- Otherwise, ignore the event.

3.6.7.9 Create Resource Manager

The Create Resource Manager event MUST be signaled with the following arguments:

- A resource manager object.

If the Create Resource Manager event is signaled, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- Search for a resource manager object in the transaction manager's Active Resource Manager table with the same resource manager identifier as the provided resource manager object.

- If such a resource manager object is found in the table:
 - Send a [TXUSER_RESOURCEMANAGER_MTAG_DUPLICATE](#) message using the connection object that is referenced by the provided resource manager object.
 - Set the state of the connection object that referenced the provided resource manager object to Ended.
 - If the connection object that is referenced by the found resource manager object is of type [CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL](#) and the state of the connection is either Reenlisting or Active:
 - Send a [TXUSER_RESOURCEMANAGERINTERNAL_MTAG_DUPLICATEDETECTED](#) message using the connection object that is referenced by the found resource manager object.
- Otherwise
 - Add the provided enlistment object to the Active Resource Manager table, using the resource manager identifier field as the key.
 - Send a [TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE](#) (section 2.2.10.1.1.4) message using the connection.
 - Set the connection state to Reenlisting.

3.6.7.10 Create Subordinate Enlistment Failure

The Create Subordinate Enlistment Failure event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:
 - Log Full
 - Too Late
 - Too Many

If the Create Subordinate Enlistment Failure event is signaled, the transaction manager MUST perform the following actions:

- If the connection state is Processing Enlistment Request:
 - Send the matching message for the following reason codes using the connection of the provided enlistment:
 - Log Full: [TXUSER_ENLISTMENT_MTAG_ENLIST_LOG_FULL](#).
 - Too Late: [TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_LATE](#).
 - Too Many: [TXUSER_ENLISTMENT_MTAG_ENLIST_TOO_MANY](#).
 - Set the connection state to [Ended](#).
- Otherwise, ignore the event.

3.6.7.11 Create Subordinate Enlistment Success

The Create Subordinate Enlistment Success event MUST be signaled with the following arguments:

- An Enlistment object.

If the Create Subordinate Enlistment Success event is signaled, the transaction manager MUST perform the following actions:

- If the connection state is Processing Enlistment Request:
 - Send a [TXUSER ENLISTMENT MTAG ENLISTED](#) message using the connection of the provided enlistment.
 - Set the connection state to Active.
- Otherwise, ignore the event.

3.6.7.12 Create Voter Enlistment Failure

The Create Voter Enlistment Failure event MUST be signaled with the following arguments:

- An enlistment object.
- A value indicating the failure reason. The reason MUST be set to the following value:
 - Too Late

If the Create Voter Enlistment Failure event is signaled, the transaction manager MUST perform the following actions:

- If the connection state is Creating Voter:
 - Send the [TXUSER VOTER MTAG CREATE TOO LATE \(section 2.2.10.4.1.5\)](#) message using the connection of the provided enlistment:
 - Set the connection state to Ended.
- Otherwise, ignore the event.

3.6.7.13 Create Voter Enlistment Success

The Create Voter Enlistment Success event MUST be signaled with the following arguments:

- An enlistment object.

If the Create Voter Enlistment Success event is signaled, the Transaction Manager MUST perform the following actions:

- If the connection state is Creating Voter:
 - Send a [TXUSER VOTER MTAG CREATED \(section 2.2.10.4.1.7\)](#) message using the connection of the provided enlistment.
 - Set the connection state to Active.
- Otherwise, ignore the event.

3.6.7.14 Phase Zero Aborted

The Phase Zero Aborted event MUST be signaled with the following arguments:

- An enlistment object.

If the Phase Zero Aborted event is signaled, the transaction manager MUST perform the following actions:

- If the connection state is Active:
 - Send a [TXUSER PHASE0 MTAG PHASE0REQ ABORT](#) message using the connection of the provided enlistment.
 - Set the connection state to ended.
- Otherwise, ignore the event.

3.6.7.15 Reenlist Complete

The Reenlist Complete event MUST be signaled with the following arguments:

- A resource manager object.

If the Reenlist Complete event is signaled, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- For each enlistment in the [Failed to Notify](#) table:
 - If the resource manager identifier field of the Enlistment object matches the provided resource manager object's identifier:
 - Signal the [Enlistment Commit Complete](#) event on the [Core Transaction Manager Facet](#) providing the Enlistment object.
 - Remove the Enlistment object from the Failed to Notify list.
- Send a [TXUSER RESOURCEMANAGER MTAG REQUEST COMPLETE](#) message using the connection of the provided resource manager.

3.6.7.16 Resource Manager Down

The Resource Manager Down event MUST be signaled with the following arguments:

- A resource manager object.

If the Resource Manager Down event is signaled, the transaction manager communicating with a resource manager facet MUST perform the following actions:

- For each enlistment in the Failed to Notify list:
 - If the Enlistment object's resource manager identifier field matches the provided resource manager object's identifier:
 - Set the state of the connection object referenced by the Enlistment object to [Ended](#).
- Search for a resource manager object in the manager's Active Resource Manager table with the same resource manager identifier as the provided resource manager object.

- If such a resource manager object is found in the table, remove the resource manager object from the table.

3.7 Superior Transaction Manager Facet Details

3.7.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Note that the abstract data model can be implemented in a variety of ways. This protocol does not prescribe or advocate any specific implementation technique.

The superior transaction manager facet **MUST** maintain all the data elements that are specified in section [3.2.1.3](#).

The Superior Transaction Manager facet **MUST** also maintain the following data elements:

Enlistment objects that are created by the superior transaction manager facet **MUST** provide the following properties as specified in [3.2.1.3](#):

- **Name:** The Hostname field in the enlistment object's connection object.
- **Identifier:** An empty string.

The superior transaction manager **MUST** provide the states that are defined in the following sections for its supported connection types. [Connection Types Relevant to Resource Managers - Versioning](#) defines the connection types that a superior transaction manager **MUST** provide for each supported protocol version.

3.7.1.1 CONNTYPE_PARTNERTM_PROPAGATE Initiator States

The superior transaction manager **MUST** act as an initiator for the [CONNTYPE PARTNERTM PROPAGATE \(section 2.2.9.1.1.1\)](#) connection type. In this role, the superior transaction manager **MUST** provide support for the states in this section.

3.7.1.1.1 Idle

This is the initial state. The following events are processed in this state:

- [Propagate Transaction \(section 3.7.7.10\)](#).

3.7.1.1.2 Awaiting Propagation Response

The following events are processed in this state:

- [Receiving a PARTNERTM_PROPAGATE_MTAG_PROPAGATED Message \(section 3.7.5.1.1.1.1\)](#).
- [Receiving a PARTNERTM_PROPAGATE_MTAG_DUPLICATE, PARTNERTM_PROPAGATE_MTAG_NO_MEM, or PARTNERTM_PROPAGATE_MTAG_LOG_FULL Message \(section 3.7.5.1.1.1.2\)](#).

3.7.1.1.3 Active

The following events are processed in this state:

- [Receiving PARTNERTM PROPAGATE MTAG PHASE0REGISTER \(section 3.7.5.1.2.1.2\).](#)
- [Receiving PARTNERTM PROPAGATE MTAG ABORTNOTIFY \(section 3.7.5.1.2.1.4\).](#)
- [Begin Phase One \(section 3.6.7.3\).](#)
- [Begin Rollback \(section 3.6.7.5\).](#)

3.7.1.1.4 Awaiting Abort Response

The following events are processed in this state:

- [Receiving PARTNERTM PROPAGATE MTAG ABORTREQDONE \(section 3.7.5.1.2.1.5\)](#)

3.7.1.1.5 Phase Zero Registration

The following events are processed in this state:

- [Create Phase Zero Enlistment Success \(section 3.7.7.6\).](#)
- [Create Phase Zero Enlistment Failure \(section 3.7.7.5\).](#)

3.7.1.1.6 Requesting Phase Zero

The following events are processed in this state:

- [Receiving PARTNERTM PROPAGATE MTAG ABORTNOTIFY \(section 3.7.5.1.2.1.4\).](#)
- [Begin Phase Zero \(section 3.6.7.4\).](#)
- [Receiving PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 3.7.5.1.2.1.8\).](#)

3.7.1.1.7 Phase Zero

The following events are processed in this state:

- [Receiving PARTNERTM PROPAGATE MTAG PHASE0COMPLETE \(section 3.7.5.1.2.1.3\).](#)
- [Receiving PARTNERTM PROPAGATE MTAG PHASE0REGISTER \(section 3.7.5.1.2.1.2\).](#)
- [Receiving PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 3.7.5.1.2.1.8\).](#)

3.7.1.1.8 Phase Zero Registration During Phase Zero

The following events are processed in this state:

- [Create Phase Zero Enlistment Success \(section 3.7.7.6\).](#)
- [Create Phase Zero Enlistment Failure \(section 3.7.7.5\).](#)

3.7.1.1.9 Phase Zero with Outstanding Registration

The following events are processed in this state:

- [Receiving PARTNERTM PROPAGATE MTAG PHASE0COMPLETE \(section 3.7.5.1.2.1.3\).](#)
- [Receiving PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 3.7.5.1.2.1.8\)](#)

3.7.1.1.10 Awaiting Prepare Response

The following events are processed in this state:

- [Receiving PARTNERTM PROPAGATE MTAG PREPAREREQDONE \(section 3.7.5.1.2.1.6\).](#)
- [Receiving PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 3.7.5.1.2.1.8\).](#)

3.7.1.1.11 Prepared

The following events are processed in this state:

- [Begin Commit \(section 3.7.7.1\).](#)
- [Begin Rollback \(section 3.7.7.4\).](#)

3.7.1.1.12 Awaiting Commit Response

The following events are processed in this state:

- [Receiving a PARTNERTM PROPAGATE MTAG COMMITREQDONE \(section 3.7.5.1.2.1.7\).](#)
- [Receiving a PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 3.7.5.1.2.1.8\).](#)

3.7.1.1.13 Ended

This is the final state.

3.7.1.1.14 State Diagram

The following illustration shows the relationship between the [CONNTYPE PARTNERTM PROPAGATE](#) initiator states.

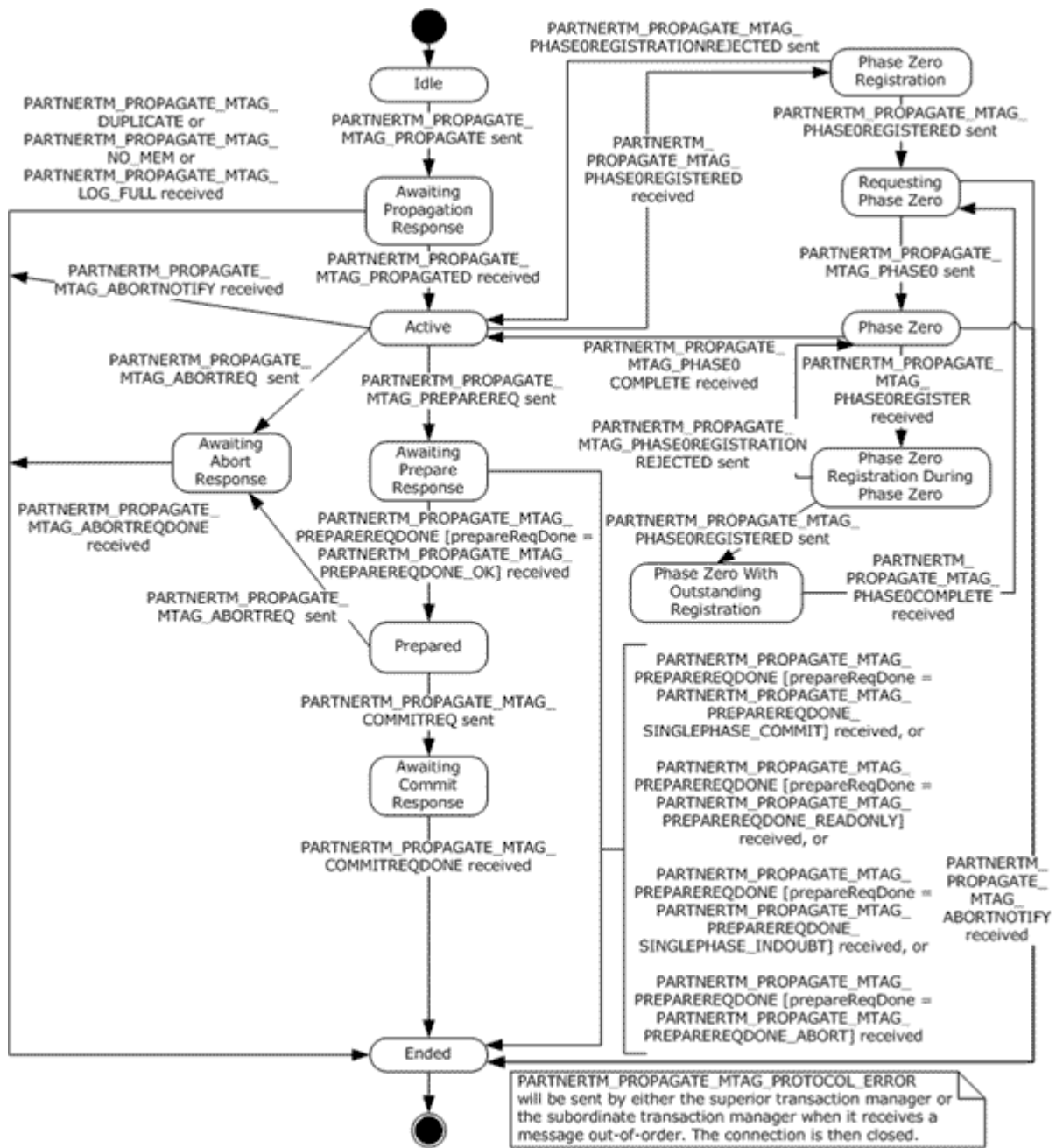


Figure 55: CONNTYPE_PARTNERTM_PROPAGATE initiator states

3.7.1.2 CONNTYPE_PARTNERTM_BRANCH Acceptor States

The superior transaction manager MUST act as an acceptor for the [CONNTYPE PARTNERTM BRANCH \(section 2.2.9.1.2.1\)](#) connection type. In this role, the superior transaction manager MUST provide support for the states in this section.

3.7.1.2.1 Idle

This is the initial state. The following events are processed in this state:

- Receiving [PARTNERTM_BRANCH_MTAG_BRANCHING \(section 3.7.5.1.2.1.1\)](#).

3.7.1.2.2 Branching

The following events are processed in this state:

- [Create Subordinate Enlistment Success \(section 3.6.7.11\)](#).
- [Create Subordinate Enlistment Failure \(section 3.7.7.7\)](#).

3.7.1.2.3 Active

The following events are processed in this state:

- Receiving [PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER \(section 2.2.9.1.1.1.14\)](#).
- Receiving [PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY \(section 2.2.9.1.1.1.13\)](#).
- [Begin Phase One \(section 3.6.7.3\)](#).
- [Begin Rollback \(section 3.4.7.5\)](#).

3.7.1.2.4 Awaiting Abort Response

The following events are processed in this state:

- Receiving [PARTNERTM_PROPAGATE_MTAG_ABORTREQDONE \(section 2.2.9.1.1.1.12\)](#).

3.7.1.2.5 Phase Zero Registration

The following events are processed in this state:

- [Create Phase Zero Enlistment Success \(section 3.7.7.6\)](#).
- [Create Phase Zero Enlistment Failure \(section 3.7.7.5\)](#).

3.7.1.2.6 Requesting Phase Zero

The following events are processed in this state:

- [Receiving PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY \(section 3.7.5.1.2.1.4\)](#).
- [Begin Phase Zero \(section 3.7.7.3\)](#).
- [Receiving PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR \(section 3.7.5.1.2.1.8\)](#).

3.7.1.2.7 Phase Zero

The following events are processed in this state:

- [Receiving PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE \(section 3.7.5.1.2.1.3\)](#).
- [Receiving PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER \(section 3.7.5.1.2.1.2\)](#).

- [Receiving PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 3.7.5.1.2.1.8\).](#)

3.7.1.2.8 Phase Zero Registration During Phase Zero

The following events are processed in this state:

- [Create Phase Zero Enlistment Success \(section 3.7.7.6\).](#)
- [Create Phase Zero Enlistment Failure \(section 3.7.7.5\).](#)

3.7.1.2.9 Phase Zero with Outstanding Registration

The following events are processed in this state:

- [Receiving PARTNERTM PROPAGATE MTAG PHASE0COMPLETE \(section 3.7.5.1.2.1.3\).](#)
- [Receiving PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 3.7.5.1.2.1.8\).](#)

3.7.1.2.10 Awaiting Prepare Response

The following events are processed in this state:

- [Receiving PARTNERTM PROPAGATE MTAG PREPAREREQDONE \(section 3.7.5.1.2.1.6\)](#)
- [Receiving PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 3.7.5.1.2.1.8\)](#)

3.7.1.2.11 Prepared

The following events are processed in this state:

- [Begin Commit \(section 3.7.7.1\).](#)
- [Begin Rollback \(section 3.7.7.4\).](#)

3.7.1.2.12 Awaiting Commit Response

The following events are processed in this state:

- [Receiving a PARTNERTM PROPAGATE MTAG COMMITREQDONE \(section 3.7.5.1.2.1.7\).](#)
- [Receiving a PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 3.7.5.1.1.1.4\).](#)

3.7.1.2.13 Ended

This is the final state.

3.7.1.2.14 State Diagram

The following figure shows the relationship between the [CONNTYPE PARTNERTM BRANCH](#) acceptor states.

3.7.1.3.1 Idle

This is the initial state. The following events are processed in this state:

- [Begin Commit \(section 3.7.7.1\)](#).

3.7.1.3.2 Awaiting Confirmation

The following events are processed in this state:

- [Receiving a PARTNERTIM_REDELIVERCOMMIT_MTAG_COMMITREQDONE \(section 3.7.5.1.2.1.7\)](#).
- [Receiving a PARTNERTIM_REDELIVERCOMMIT_MTAG_RETRY Message \(section 3.7.5.2.2.1.2\)](#).

3.7.1.3.3 Waiting to Rerequest

The following events are processed in this state:

- [Redeliver Commit Timer \(section 3.7.2.1\)](#).

3.7.1.3.4 Ended

This is the final state.

3.7.1.3.5 State Diagram

The following figure shows the relationship between the [CONNTYPE_PARTNERTM_REDELIVERCOMMIT](#) initiator states.

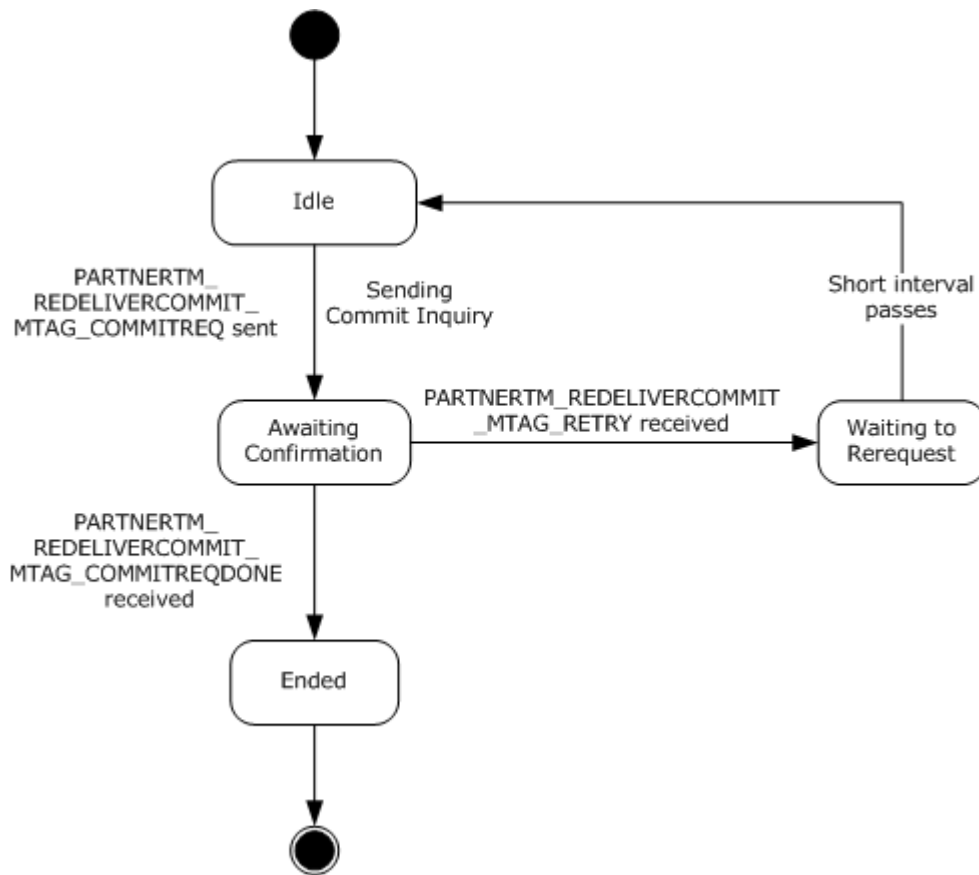


Figure 57: CONNTYPE_PARTNERTM_REDELIVERCOMMIT initiator states

3.7.1.4 CONNTYPE_PARTNERTM_CHECKABORT Acceptor States

The superior transaction manager MUST act as an acceptor for the [CONNTYPE PARTNERTM CHECKABORT \(section 2.2.9.2.1.1\)](#) connection type. In this role, the superior transaction manager MUST provide support for the states in this section.

3.7.1.4.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a PARTNERTM CHECKABORT MTAG CHECK Message \(section 3.7.5.2.1.1.1\)](#).

3.7.1.4.2 Processing Abort Inquiry

This is a transient state that is assumed during the processing of a request for check abort. No specific events are processed in this state.

3.7.1.4.3 Ended

This is the final state.

3.7.1.4.4 State Diagram

The following figure shows the relationship between the [CONNTYPE_PARTNERTM_CHECKABORT](#) acceptor states.

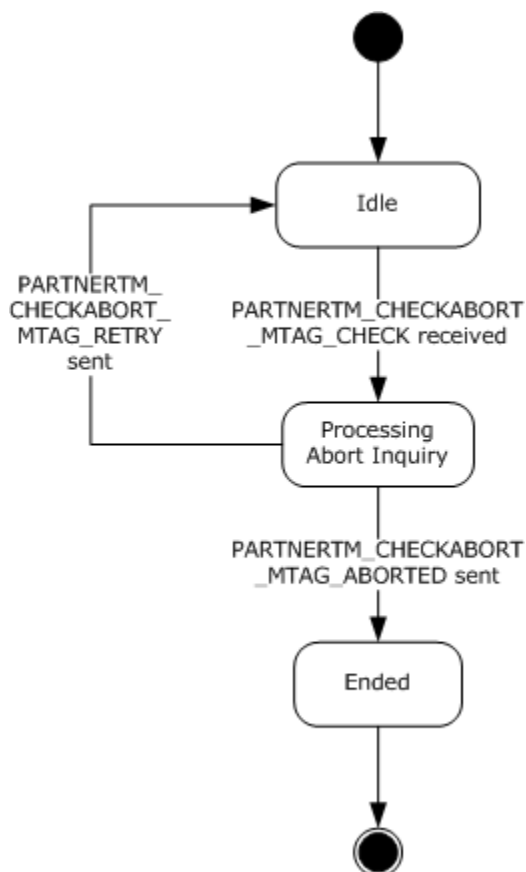


Figure 58: CONNTYPE_PARTNERTM_CHECKABORT acceptor states

3.7.2 Timers

The superior transaction manager facet MUST provide the following [Redeliver Commit Timer](#).

3.7.2.1 Redeliver Commit Timer

This timer MUST be set when the [Superior Transaction Manager Facet \(section 1.3.3.3.4\)](#) receives a [PARTNERTM_REDELIVERCOMMIT_MTAG_RETRY \(section 2.2.9.2.2.1.3\)](#) message on a [CONNTYPE_PARTNERTM_REDELIVERCOMMIT \(section 2.2.9.2.2.1\)](#) connection. The timer MUST be canceled when the `CONNTYPE_PARTNERTM_REDELIVERCOMMIT` (section 2.2.9.2.2.1) connection is disconnected.

The default, minimum, and maximum values of the timer are implementation-specific. [<50>](#)

When the timer is initialized, the Superior Transaction Manager Facet MUST provide an Enlistment object to associate with the timer. When the timer expires, the same Enlistment object MUST be provided alongside the timer notification. The Superior Transaction Manager Facet MUST provide a

distinct [Reenlist Timeout Timer \(section 3.6.6.1\)](#) instance for each CONNTYPE_PARTNERTM_REDELIVERCOMMIT connection.

3.7.3 Initialization

When the superior transaction manager facet is initialized:

- The superior transaction managerfacet MUST examine the following security flags on the core transaction manager and perform the following actions:
 - If one of the Allow Network Access, Allow Network Transactions, or Allow Outbound Transactions flags is set to false:
 - The superior transaction manager MUST refuse to accept incoming connections from remote machines for the following connection types:
 - [CONNTYPE PARTNERTM BRANCH \(section 2.2.9.1.2.1\)](#)
 - If one of the Allow Network Access or Allow Network Transactions flags is set to false, or if both the Allow Inbound Transactions and Allow Outbound Transactions flags are set to false:
 - The superior transaction manager MUST refuse to accept incoming connections from remote machines for the following connection types:
 - [CONNTYPE PARTNERTM CHECKABORT \(section 2.2.9.2.1.1\)](#)

3.7.4 Higher-Layer Triggered Events

No higher-layer triggered events apply.

3.7.5 Message Processing Events and Sequencing Rules

3.7.5.1 Transaction Propagation and Coordination

3.7.5.1.1 Push Propagation

3.7.5.1.1.1 CONNTYPE_PARTNERTM_PROPAGATE as Initiator

For all messages that are received in this connection type, the superior transaction manager MUST process the message as specified in section [3.1](#).

Also, for incoming messages, the superior transaction manager MUST override the verification actions of the default state as specified in section [3.1](#) in the following manner:

- If the current connection state does not define a processing rule for the message:
 - Send a [PARTNERTM_PROPAGATE MTAG_PROTOCOL_ERROR](#) message using the connection.
 - Perform default invalid message processing, as specified in section [3.1](#).

The superior transaction manager MUST also follow the processing rules that are specified in the following sections.

3.7.5.1.1.1.1 Receiving a PARTNERTM_PROPAGATE_MTAG_PROPAGATED Message

When the superior transaction manager receives a [PARTNERTM_PROPAGATE_MTAG_PROPAGATED \(section 2.2.9.1.1.1.1\)](#) message, the superior transaction manager MUST perform the following actions:

- If the connection state is Awaiting Propagation Response:
 - Set the connection state to [Active \(section 3.2.1.4.2\)](#).
 - Create an Enlistment object with the following values:
 - The [Superior Transaction Manager Facet \(section 1.3.3.3.4\)](#).
 - The transaction object referenced by this connection.
 - This connection object.
 - Signal the [Propagate Transaction Success \(section 3.2.7.27\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the created enlistment object.
- Otherwise, the message MUST be processed as specified in section [3.1](#).

3.7.5.1.1.1.2 Receiving a PARTNERTM_PROPAGATE_MTAG_DUPLICATE, PARTNERTM_PROPAGATE_MTAG_NO_MEM, or PARTNERTM_PROPAGATE_MTAG_LOG_FULL Message

When the [Superior Transaction Manager Facet](#) receives a [PARTNERTM_PROPAGATE_MTAG_DUPLICATE](#), [PARTNERTM_PROPAGATE_MTAG_NO_MEM](#), or [PARTNERTM_PROPAGATE_MTAG_LOG_FULL](#) message, the superior transaction manager MUST perform the following actions:

- If the connection state is Awaiting Propagation Response:
 - Create an Enlistment object with the following values:
 - The Superior Transaction Manager Facet.
 - The transaction object that is referenced by this connection.
 - This connection object.
 - Signal the [Propagate Transaction Failure](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The created Enlistment object.
 - The failure code that matches the incoming message:
 - PARTNERTM_PROPAGATE_MTAG_DUPLICATE: Duplicate.
 - PARTNERTM_PROPAGATE_MTAG_NO_MEM: No Mem.
 - PARTNERTM_PROPAGATE_MTAG_LOG_FULL: Log Full.
- Set the connection state to [Ended](#).

- Otherwise, the message MUST be processed as specified in section [3.1](#).

3.7.5.1.1.1.3 Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER, PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE, PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE, PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE, PARTNERTM_PROPAGATE_MTAG_ABORTREQDONE, or PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY Message

When the superior transaction manager facet receives one of these messages, it MUST follow the same message processing rules as the [CONNTYPE PARTNERTM BRANCH](#) connection type acting as an acceptor. See section [3.7.5.1.2.1](#) for more information.

3.7.5.1.1.1.4 Receiving a PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR Message

The processing of this event MUST be identical to the processing of the [Connection Disconnected](#) event.

3.7.5.1.1.1.5 Connection Disconnected

When a [CONNTYPE PARTNERTM PROPAGATE \(section 2.2.9.1.1.1\)](#) connection is disconnected, the application MUST perform the same actions as the [CONNTYPE PARTNERTM BRANCH \(section 2.2.9.1.2.1\)](#) connection type acting as an acceptor. For more information, see section [3.7.5.1.2.1](#).

3.7.5.1.2 Pull Propagation

3.7.5.1.2.1 CONNTYPE_PARTNERTM_BRANCH as Acceptor

For all messages that are received in this connection type, the superior transaction manager MUST process the message as specified in section [3.7.5](#).

Also, for incoming messages, the superior transaction manager MUST override the verification actions of the default state, as specified in section [3.7.5](#), in the following manner:

- If the current connection state does not define a processing rule for the message:
 - Send a [PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR](#) message using the connection.
 - Perform default invalid message processing, as specified in section [3.1.6](#).

The superior transaction manager MUST also follow the processing rules as specified in the following section.

3.7.5.1.2.1.1 Receiving a PARTNERTM_BRANCH_MTAG_BRANCHING Message

When the superior transaction manager receives a [PARTNERTM_BRANCH_MTAG_BRANCHING](#) message, the superior transaction manager MUST perform the following actions:

- If the connection state is Idle:
 - Find the transaction object in the transaction table of the transaction manager using the **guidTx** field from the message as the key.
 - If a transaction object is found:

- Create a new Enlistment object with the following values:
 - The [Superior Transaction Manager Facet](#).
 - The transaction object.
 - The connection object.
- Set the **enlistment** field of the connection to the new Enlistment object.
- Signal the [Create Subordinate Enlistment](#) event on the [Core Transaction Manager Facet](#) with the new Enlistment object.
- Set the connection state to Branching.
- Otherwise:
 - Send a [PARTNERTM_BRANCH_MTAG_BRANCH_TX_NOT_FOUND](#) message using the connection.
 - Set the connection state to [Ended](#).
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.7.5.1.2.1.2 Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER Message

When the superior transaction manager receives a [PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER \(section 2.2.9.1.1.1.14\)](#) message, the superior transaction manager MUST perform the following actions:

- If the connection state is Active:
 - Set the connection state to Phase Zero Registration.
 - Signal the [Create Phase Zero Enlistment \(section 3.2.7.10\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the Enlistment object that is referenced by this connection.
- Otherwise, if the connection state is Phase Zero:
 - Set the connection state to Phase Zero Registration During Phase Zero.
 - Signal the Create Phase Zero Enlistment event on the Core Transaction Manager Facet with the Enlistment object that is referenced by this connection.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.7.5.1.2.1.3 Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE Message

When the superior transaction manager receives a [PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE](#) message, the superior transaction manager MUST perform the following actions:

- If the connection state is Phase Zero:
 - Set the connection state to Active.

- Signal the [Enlistment Phase Zero Complete](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The Enlistment object that is referenced by this connection.
 - The Completed outcome value.
- Otherwise, if the connection state is Phase Zero With Outstanding Registration:
 - Set the connection state to Requesting Phase Zero.
 - Signal the Enlistment Phase Zero Complete event on the Core Transaction Manager Facet with the following arguments:
 - The Enlistment object that is referenced by this connection.
 - The Completed outcome value.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.7.5.1.2.1.4 Receiving a PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY Message

When the superior transaction manager receives a [PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY \(section 2.2.9.1.1.1.13\)](#) message, the superior transaction manager MUST perform the following actions:

- If the connection state is either Active or Requesting Phase Zero:
 - Signal the [Enlistment Unilaterally Aborted \(section 3.2.7.19\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the Enlistment object that is referenced by this connection.
 - Set the connection state to [Ended \(section 3.2.1.4.14\)](#).
- Otherwise, if the connection state is [Phase Zero \(section 3.2.1.4.3\)](#):
 - Signal the [Enlistment Phase Zero Complete \(section 3.2.7.17\)](#) event on the Core Transaction Manager Facet with the following arguments:
 - The Enlistment object that is referenced by this connection.
 - The Aborted outcome value.
 - Set the connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message, as specified in section [3.1.6](#).

3.7.5.1.2.1.5 Receiving a PARTNERTM_PROPAGATE_MTAG_ABORTREQDONE Message

When the superior transaction manager receives a [PARTNERTM_PROPAGATE_MTAG_ABORTREQDONE](#) message, the superior transaction manager MUST perform the following actions:

- If the connection state is Awaiting Abort Response:

- Signal the [Enlistment Rollback Complete](#) event on the [Core Transaction Manager Facet](#) with the Enlistment object that is referenced by this connection.
- Set the connection state to [Ended](#).
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.7.5.1.2.1.6 Receiving a PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE Message

When the superior transaction manager receives a [PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE](#) message, the superior transaction manager MUST perform the following actions:

- If the connection state is Awaiting Prepare Response:
 - Signal the [Enlistment Phase One Complete \(section 3.2.7.16\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the following arguments:
 - The Enlistment object that is referenced by this connection.
 - The outcome value that is determined by the **prepareReqDone** field from the message. The outcome value is set to:
 - Prepared if the **prepareReqDone** field is PARTNERTM_PROPAGATE_PREPAREREQDONE_OK.
 - Aborted if the prepareReqDone field is PARTNERTM_PROPAGATE_PREPAREREQDONE_ABORT.
 - Read Only if the prepareReqDone field is PARTNERTM_PROPAGATE_PREPAREREQDONE_READ_ONLY.
 - Committed if the prepareReqDone field is PARTNERTM_PROPAGATE_PREPAREREQDONE_SINGLEPHASE_COMMIT.
 - [In Doubt](#) if the prepareReqDone field is PARTNERTM_PROPAGATE_PREPAREREQDONE_SINGLEPHASE_INDOUBT.
 - If the prepareReqDone value is Prepared:
 - Set the connection state to Prepared.
 - Otherwise, set the connection state to [Ended](#).
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.7.5.1.2.1.7 Receiving a PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE Message

When the superior transaction manager receives a [PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE \(section 2.2.9.1.1.1.10\)](#) message, the superior transaction manager MUST perform the following actions:

- If the connection state is Awaiting Commit Response:

- Signal the [Enlistment Commit Complete \(section 3.2.7.15\)](#) event on the core transaction manager with the Enlistment object that is referenced by this connection.
- Set the connection state to [Ended \(section 3.2.1.4.14\)](#).
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.7.5.1.2.1.8 Receiving a PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR Message

The processing of this event MUST be identical to the processing of the [Connection Disconnected](#) event.

3.7.5.1.2.1.9 Connection Disconnected

When a [CONNTYPE PARTNERTM BRANCH \(section 2.2.9.1.2.1\)](#) connection is disconnected, the superior transaction manager facet MUST perform the following actions:

- If the connection state is Awaiting Prepare Response:
 - If the state of the transaction object that is referenced by the connection is [Single Phase Commit \(section 3.2.1.4.9\)](#):
 - Signal the [Enlistment Phase Zero Complete \(section 3.2.7.17\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the following arguments:
 - The Enlistment object that is referenced by this connection.
 - The In Doubt outcome value.
 - Otherwise:
 - Signal the Enlistment Phase Zero Complete event on the Core Transaction Manager Facet with the following arguments:
 - The Enlistment object that this connection references.
 - The Aborted outcome value.
- Otherwise, if the connection state is Awaiting Commit Response:
 - Retrieve the Enlistment object that is referenced by the connection object.
 - Initiate a new [CONNTYPE PARTNERTM REDELIVERCOMMIT \(section 2.2.9.2.2.1\)](#) connection using the Name object of this connection.
 - Add the new connection object to the enlistment transaction Connection list.
 - Assign the new connection object to the enlistment Connection field.
 - Assign the enlistment to the connection **Enlistment** field.
 - Send a [PARTNERTM REDELIVERCOMMIT MTAG COMMITREQ \(section 2.2.9.2.2.1.1\)](#) message using the connection:
 - Set the **guidTX** field to the identifier of the transaction object that is referenced by this connectionEnlistment object.

- Set the new connection state to Awaiting Confirmation.
- Otherwise, if the connection state is Awaiting Abort Response:
 - Signal the [Enlistment Rollback Complete \(section 3.2.7.18\)](#) event on the Core Transaction Manager Facet with the Enlistment object that is referenced by this connection.
- Otherwise, if the connection state is either Phase Zero, Phase Zero Registration During Phase Zero, or Phase Zero With Outstanding Registration:
 - Signal the Enlistment Phase Zero Complete event on the Core Transaction Manager Facet with the following arguments:
 - The Enlistment object that is referenced by this connection.
 - The Aborted outcome value.
- Otherwise, if the connection state is either Branching, Active, Phase Zero Registration, or Requesting Phase Zero:
 - Signal the [Enlistment Unilaterally Aborted \(section 3.2.7.19\)](#) event on the Core Transaction Manager Facet with the Enlistment object that is referenced by this connection.
- Otherwise, the event MUST be processed as specified in section [3.1.6](#).

3.7.5.2 Transaction Recovery

3.7.5.2.1 Subordinate-Driven Recovery

3.7.5.2.1.1 CONNTYPE_PARTNERTM_CHECKABORT as Acceptor

For all messages received in this connection type, the Superior Transaction Manager facet MUST process the message in accordance with section [3.1](#). The Superior Transaction Manager facet MUST additionally follow the processing rules specified in the following sections.

3.7.5.2.1.1.1 Receiving a PARTNERTM_CHECKABORT_MTAG_CHECK Message

When the Superior Transaction Manager Facet receives a [PARTNERTM_CHECKABORT_MTAG_CHECK](#) message, the Superior Transaction Manager Facet MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Abort Inquiry.
 - Find the Transaction object in the Transaction Manager's Transaction Table, using the **guidTx** field from the message as a key.
 - If the Transaction object is not found, or if the Transaction state is either Aborting or Ended:
 - Send a [PARTNERTM_CHECKABORT_MTAG_ABORTED](#) message using the connection.
 - Set the connection state to Ended.
- Otherwise:
 - Send a [PARTNERTM_CHECKABORT_MTAG_RETRY](#) message using the connection.

- Set the connection state to Idle.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.7.5.2.1.1.2 Connection Disconnected

When a [CONNTYPE PARTNERTM CHECKABORT \(section 2.2.9.2.1.1\)](#) connection is disconnected, the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) MUST perform the actions as specified in section [3.7.1](#).

3.7.5.2.2 Superior-Driven Recovery

3.7.5.2.2.1 CONNTYPE_PARTNERTM_REDELIVERCOMMIT as Initiator

For all messages received in this connection type, the Superior Transaction Manager Facet MUST process the message as specified in section [1.3.3.3.4](#). The Superior Transaction Manager Facet MUST additionally follow the processing rules as specified in this section.

3.7.5.2.2.1.1 Receiving a PARTNERTIM_REDELIVERCOMMIT_MTAG_COMMITREQDONE Message

When the [Superior Transaction Manager Facet \(section 1.3.3.3.4\)](#) receives a [PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQDONE \(section 2.2.9.2.2.1.2\)](#) message, the Superior Transaction Manager Facet MUST perform the following actions:

- If the connection state is Awaiting Confirmation:
 - Signal the [Enlistment Commit Complete \(section 3.2.7.15\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the Enlistment object referenced by this connection.
 - Set the connection state to [Ended \(section 3.2.1.4.14\)](#).
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.7.5.2.2.1.2 Receiving a PARTNERTIM_REDELIVERCOMMIT_MTAG_RETRY Message

When the superior transaction manager facet receives a [PARTNERTM_REDELIVERCOMMIT_MTAG_RETRY](#) message, the superior transaction manager facet MUST perform the following actions:

- If the connection state is Awaiting Confirmation:
 - Set the connection state to Idle.
 - Initialize a Redeliver Commit timer with the following arguments:
 - The enlistment object of the connection.
 - An implementation-specific time-out value, as specified in [Redeliver Commit Timer](#).
- Otherwise, the message MUST be processed as an invalid message, as specified in section [3.1.6](#).

3.7.5.2.2.1.3 Connection Disconnected

When a [CONNTYPE PARTNERTM REDELIVERCOMMIT \(section 2.2.9.2.2.1\)](#) connection is disconnected, the [Superior Transaction Manager Facet \(section 1.3.3.3.4\)](#) MUST perform the following actions:

- If the connection state is Waiting to Rerequest:
 - Cancel the Redeliver Commit Timer associated with the connection.
- If the connection state is Idle, Waiting to Rerequest or Awaiting Confirmation:
 - Set the connection state to Ended.
 - Signal the [Begin Commit \(section 3.7.7.1\)](#) event on the Superior Transaction Manager Facet with the provided Enlistment object.
- Otherwise, the event MUST be processed as specified in section [3.1.6](#).

3.7.6 Timer Events

3.7.6.1 Redeliver Commit Timer

When this timer expires, the [Superior Transaction Manager Facet \(section 1.3.3.3.4\)](#) MUST perform the following actions:

- Cancel the Redeliver Commit timer.
- If the connection referenced by the provided Enlistment is in the Waiting to Rerequest state:
 - Send a [PARTNERTM REDELIVERCOMMIT MTAG COMMITREQ \(section 2.2.9.2.2.1.1\)](#) message using the Connection referenced by the provided Enlistment object:
 - Set the **guidTX** field to the provided Enlistment object's Transaction object's Identifier field.
 - Set the connection state to Awaiting Confirmation.
- Otherwise, ignore the timer event.

3.7.7 Other Local Events

The Superior Transaction Manager MUST be prepared to process the local events defined in the following sections.

The Superior Transaction Manager MUST be prepared to process local events pertaining to [Phase Zero](#) functionality only on versions where the connection type [CONNTYPE TXUSER PHASE0](#) is supported. [Connection Types Relevant to Resource Managers - Versioning](#) defines protocol version support for this connection type. The following local events are affected:

- [Create Phase Zero Enlistment Success](#)
- [Create Phase Zero Enlistment Failure](#)
- [Begin Phase Zero](#)
- [Phase Zero Aborted](#)

3.7.7.1 Begin Commit

The Begin Commit event MUST be signaled with the following arguments:

- An Enlistment object.

If the Begin Commit event is signaled, the [Superior Transaction Manager Facet \(section 1.3.3.3.4\)](#) MUST perform the following actions:

- If the connection state is Ended:
 - Initiate a new [CONNTYPE PARTNERTM REDELIVERCOMMIT \(section 2.2.9.2.2.1\)](#) connection to the provided enlistment's Name object.
 - Add the new connection to the provided enlistment's transaction's connection list.
 - Set the new connection to the enlistment's Connection field.
 - Set the Enlistment to the connection's Enlistment field.
 - Send a [PARTNERTM REDELIVERCOMMIT MTAG COMMITREQ \(section 2.2.9.2.2.1.1\)](#) message using the new connection.
 - Set the **guidTx** field to the Identifier field of the Transaction object referenced by this connection's Enlistment object.
 - Set the new connection state to Awaiting Confirmation.
- Otherwise:
 - Send a PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQ (section 2.2.9.2.2.1.1) message using the connection.
 - Set the connection state to Awaiting Commit Response.

3.7.7.2 Begin Phase One

The Begin Phase One event MUST be signaled with the following arguments:

- An Enlistment object.
- A Boolean Single-Phase Commit value:
 - If true, the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) SHOULD attempt to perform a Single-Phase Commit.
 - If false, the Subordinate Transaction Manager Facet MUST NOT attempt to perform a Single-Phase Commit.

If the Begin Phase One event is signaled, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the provided Single-Phase Commit value is set to true:
 - Send a [PARTNERTM PROPAGATE MTAG PREPAREREQ \(section 2.2.9.1.1.1.6\)](#) message using the connection.
 - Set the **fSinglePhase** field to a nonzero value.

- Otherwise:
 - Send a PARTNERTM_PROPAGATE_MTAG_PREPAREREQ (section 2.2.9.1.1.1.6) message using the connection.
 - Set the **fSinglePhase** field to zero.
- Set the connection state to Awaiting Prepare Response.

3.7.7.3 Begin Phase Zero

The Begin Phase Zero event MUST be signaled with the following arguments:

- An Enlistment object.

If the Begin Phase Zero event is signaled, the [Superior Transaction Manager Facet \(section 1.3.3.3.4\)](#) MUST perform the following actions:

- Send a [PARTNERTM_PROPAGATE_MTAG_PHASE0 \(section 2.2.9.1.1.1.17\)](#) message using the connection.
- Set the connection state to Phase Zero.

3.7.7.4 Begin Rollback

The Begin Rollback event MUST be signaled with the following arguments:

- An Enlistment object.

If the Begin Rollback event is signaled, the [Superior Transaction Manager Facet](#) MUST perform the following actions:

- If the provided Enlistment's connection state is Ended:
 - Signal the [Enlistment Rollback Complete](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The provided Enlistment object.
- Otherwise:
 - Send a [PARTNERTM_PROPAGATE_MTAG_ABORTREQ](#) message using the connection.
 - Set the connection state to Awaiting Abort Response.

3.7.7.5 Create Phase Zero Enlistment Failure

The Create Phase Zero Enlistment Failure event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:
 - Too Late
 - Tx Not Found

If the Create Phase Zero Enlistment Failure event is signaled, the [Superior Transaction Manager Facet \(section 1.3.3.3.4\)](#) MUST perform the following actions:

- Send a [PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTRATIONREJECTED \(section 2.2.9.1.1.1.16\)](#) message using the enlistment's connection.
- If the enlistment's connection state is Phase Zero Registration:
 - Set the connection state to Active.
- Otherwise, if the connection state is Phase Zero Registration During Phase Zero:
 - Set the connection state to [Phase Zero \(section 3.2.1.4.3\)](#).
- Otherwise, ignore the event.

3.7.7.6 Create Phase Zero Enlistment Success

The Create Phase Zero Enlistment Success event MUST be signaled with the following arguments:

- An Enlistment object.

If the Create Phase Zero Enlistment Success event is signaled, the [Superior Transaction Manager Facet \(section 1.3.3.3.4\)](#) MUST perform the following actions:

- Send a [PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTERED \(section 2.2.9.1.1.1.15\)](#) message using the enlistment's connection.
- If the enlistment's connection state is Phase Zero Registration:
 - Set the connection state to Requesting Phase Zero.
- Otherwise, if the connection state is Phase Zero Registration During Phase Zero:
 - Set the connection state to Phase Zero With Outstanding Registration.
- Otherwise, ignore the event.

3.7.7.7 Create Subordinate Enlistment Failure

The Create Subordinate Enlistment Failure event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:
 - Log Full
 - Too Late
 - Too Many

If the Create Subordinate Enlistment Failure event is signaled, the [Superior Transaction Manager Facet](#) MUST perform the following actions:

- Send the matching message for the following reason codes
 - Log Full: [PARTNERTM_BRANCH_MTAG_BRANCH_LOG_FULL](#).

- Too Late: [PARTNERTM_BRANCH_MTAG_BRANCH_TOO_LATE](#).
- Too Many: [PARTNERTM_BRANCH_MTAG_BRANCH_TOO_MANY](#).
- Set the connection state to Ended.

3.7.7.8 Create Subordinate Enlistment Success

The Create Subordinate Enlistment Success event MUST be signaled with the following arguments:

- An Enlistment object.

If the Create Subordinate Enlistment Success event is signaled, the [Superior Transaction Manager Facet](#) MUST perform the following actions:

- Send a [PARTNERTM_BRANCH_MTAG_BRANCHED](#) message using the enlistment's connection.
- Set the connection state to Active.

3.7.7.9 Phase Zero Aborted

The Phase Zero Aborted event MUST be signaled with the following arguments

- An Enlistment object.

If the Phase Zero Aborted event is signaled, the [Superior Transaction Manager Facet](#) MUST perform the following actions:

- Ignore the event.

3.7.7.10 Propagate Transaction

The Propagate Transaction event MUST be signaled with the following arguments:

- A Transaction object
- A Name object representing the remote subordinate transaction manager

If the Propagate Transaction event is signaled, the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) MUST perform the following actions:

- Initiate a new [CONNTYPE_PARTNERTM_PROPAGATE \(section 2.2.9.1.1.1\)](#) connection to the provided Name object.
- Add the connection to the provided transaction's connection list.
- Send a [PARTNERTM_PROPAGATE_MTAG_PROPAGATE \(section 2.2.9.1.1.1\)](#) message using the connection:
 - Set the **guidTX** field to the provided transaction's identifier.
 - Set the **isoLevel** field to the provided transaction's isolation level.
 - Set the **isoFlags** field to the provided transaction's isolation flags.
 - Set the **szDesc** field to the provided transaction's description.
- Set the connection state to Awaiting Propagation Response.

3.8 Subordinate Transaction Manager Facet Details

3.8.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Note that the abstract data model can be implemented in a variety of ways. This protocol does not prescribe or advocate any specific implementation technique.

The [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST maintain all the data elements as specified in section [3.1.1](#) and section [3.2.1](#).

Enlistment objects that are created by the subordinate transaction manager MUST provide the following properties as specified in section [3.2.1.3](#):

- **Name:** The Hostname field of the Enlistment object's Connection object.
- **Identifier:** An empty string.

The subordinate transaction manager MUST provide the states as specified in the following sections for its supported connection types. Section [2.2.1.1.2](#) defines the connection types that a subordinate transaction manager MUST provide for each supported protocol version.

3.8.1.1 CONNTYPE_PARTNERTM_PROPAGATE Acceptor States

The [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) MUST act as an acceptor for the [CONNTYPE PARTNERTM PROPAGATE \(section 2.2.9.1.1.1\)](#) connection type. In this role, the subordinate transaction manager MUST provide support for the states in this section.

3.8.1.1.1 Idle

This is the initial state. The following events are processed in this state:

- Receiving a [PARTNERTM PROPAGATE MTAG PROPAGATE message \(section 3.8.5.1.1.1.1\)](#).

3.8.1.1.2 Propagating

The following events are processed in this state:

- [Create Superior Enlistment Success \(section 3.8.7.4\)](#).
- [Create Superior Enlistment Failure \(section 3.8.7.5\)](#).

3.8.1.1.3 Active

The following events are processed in this state:

- [Register Phase Zero \(section 3.8.7.9\)](#).
- [Unilaterally Aborted \(section 3.4.7.22\)](#).
- [Receiving a PARTNERTM PROPAGATE MTAG ABORTREQ message \(section 3.8.5.1.2.1.5\)](#).

- [Receiving a PARTNERTM PROPAGATE MTAG PREPAREREQ message \(section 3.8.5.1.2.1.7\).](#)
- [Receiving a PARTNERTM PROPAGATE MTAG PROTOCOL ERROR message \(section 3.7.5.1.1.1.4\).](#)

3.8.1.1.4 Aborting

The following events are processed in this state:

- [Rollback complete \(section 3.4.7.17\).](#)

3.8.1.1.5 Awaiting Registration Response

The following events are processed in this state:

- [Receiving a PARTNERTM PROPAGATE MTAG PHASE0REGISTERED message \(section 3.8.5.1.2.1.3\)](#)
- [Receiving a PARTNERTM PROPAGATE MTAG PROTOCOL ERROR message \(section 3.8.5.1.2.1.9\)](#)

3.8.1.1.6 Awaiting Phase Zero

The following events are processed in this state:

- [Receiving a PARTNERTM PROPAGATE MTAG PHASE0 message \(section 3.8.5.1.2.1.6\).](#)
- [Unilaterally Aborted \(section 3.8.7.11\).](#)
- [Receiving a PARTNERTM PROPAGATE MTAG PROTOCOL ERROR message \(section 3.8.5.1.2.1.9\)](#)

3.8.1.1.7 Awaiting Phase Zero Outcome

The following events are processed in this state:

- [Phase Zero Complete \(section 3.8.7.6\).](#)

3.8.1.1.8 Awaiting Registration Response During Phase Zero

The following events are processed in this state:

- [Receiving a PARTNERTM PROPAGATE MTAG PHASE0REGISTERED message \(section 3.8.5.1.2.1.3\).](#)
- [Receiving a PARTNERTM PROPAGATE MTAG PHASE0REGISTRATIONREJECTED message \(section 2.2.9.1.1.1.16\).](#)
- [Receiving a PARTNERTM PROPAGATE MTAG PROTOCOL ERROR message \(section 3.7.5.1.1.1.4\).](#)

3.8.1.1.9 Awaiting Phase Zero Outcome with Outstanding Registration

The following events are processed in this state:

- [Phase Zero Complete \(section 3.8.7.6\).](#)

3.8.1.1.10 Preparing

The following events are processed in this state:

- [Phase One Complete \(section 3.8.7.7\)](#).

3.8.1.1.11 Prepared

The following events are processed in this state:

- [Receiving a PARTNERTM PROPAGATE MTAG COMMITREQ message \(section 3.8.5.1.2.1.8\)](#).
- [Receiving a PARTNERTM PROPAGATE MTAG ABORTREQ message \(section 3.8.5.1.2.1.5\)](#).
- [Receiving a PARTNERTM PROPAGATE MTAG PROTOCOL ERROR message \(section 3.8.5.1.1.1.3\)](#).

3.8.1.1.12 Committing

The following events are processed in this state:

- [Commit Complete \(section 3.8.7.3\)](#).

3.8.1.1.13 Ended

This is the final state.

3.8.1.1.14 State Diagram

The following illustration shows the relationship between the [CONNTYPE PARTNERTM PROPAGATE](#) acceptor states.

3.8.1.2.1 Idle

This is the initial state. The following events are processed in this state:

- [Branch Transaction \(section 3.8.7.1\)](#).

3.8.1.2.2 Awaiting Branch Response

The following events are processed in this state:

- Receiving a [PARTNERTM BRANCH MTAG BRANCHED \(section 2.2.9.1.2.1.6\)](#) message.
- [Receiving a PARTNERTM BRANCH MTAG BRANCH LOG FULL, PARTNERTM BRANCH MTAG BRANCH NO MEM, PARTNERTM BRANCH MTAG BRANCH TOO LATE, PARTNERTM BRANCH MTAG BRANCH TOO MANY or PARTNERTM BRANCH MTAG BRANCH TX NOT FOUND message \(section 3.8.5.1.2.1.2\)](#).

3.8.1.2.3 Active

The following events are processed in this state:

- [Register Phase Zero \(section 3.8.7.9\)](#).
- [Unilaterally Aborted \(section 3.8.7.11\)](#).
- Receiving a [PARTNERTM PROPAGATE MTAG ABORTREQ \(section 2.2.9.1.1.1.11\)](#) message.
- Receiving a [PARTNERTM PROPAGATE MTAG PREPAREREQ \(section 2.2.9.1.1.1.6\)](#) message.
- Receiving a [PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 2.2.9.1.1.1.8\)](#) message.

3.8.1.2.4 Aborting

The following events are processed in this state:

- [Rollback Complete \(section 3.4.7.17\)](#).

3.8.1.2.5 Awaiting Registration Response

The following events are processed in this state:

- Receiving a [PARTNERTM PROPAGATE MTAG PHASE0REGISTERED \(section 2.2.9.1.1.1.15\)](#) message.
- Receiving a [PARTNERTM PROPAGATE MTAG PHASE0REGISTRATIONREJECTED \(section 2.2.9.1.1.1.16\)](#) message.
- Receiving a [PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 2.2.9.1.1.1.8\)](#) message.

3.8.1.2.6 Awaiting Phase Zero

The following events are processed in this state:

- Receiving a [PARTNERTM PROPAGATE MTAG PHASE0 \(section 2.2.9.1.1.1.17\)](#) message.

- [Unilaterally Aborted \(section 3.8.7.11\)](#).
- Receiving a [PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 2.2.9.1.1.1.8\)](#) message.

3.8.1.2.7 Awaiting Phase Zero Outcome

The following events are processed in this state:

- [Phase Zero Complete \(section 3.8.7.6\)](#).

3.8.1.2.8 Awaiting Registration Response During Phase Zero

The following events are processed in this state:

- Receiving a [PARTNERTM PROPAGATE MTAG PHASE0REGISTERED \(section 2.2.9.1.1.1.15\)](#) message.
- Receiving a [PARTNERTM PROPAGATE MTAG PHASE0REGISTRATIONREJECTED \(section 2.2.9.1.1.1.16\)](#) message.
- Receiving a [PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 2.2.9.1.1.1.8\)](#) message.

3.8.1.2.9 Awaiting Phase Zero Outcome with Outstanding Registration

The following events are processed in this state:

- [Phase Zero Complete \(section 3.8.7.6\)](#).

3.8.1.2.10 Preparing

The following events are processed in this state:

- [Phase One Complete \(section 3.4.7.13\)](#).

3.8.1.2.11 Prepared

The following events are processed in this state:

- Receiving a [PARTNERTM PROPAGATE MTAG COMMITREQ \(section 2.2.9.1.1.1.9\)](#) message.
- Receiving a [PARTNERTM PROPAGATE MTAG ABORTREQ \(section 2.2.9.1.1.1.11\)](#) message.
- Receiving a [PARTNERTM PROPAGATE MTAG PROTOCOL ERROR \(section 2.2.9.1.1.1.8\)](#) message.

3.8.1.2.12 Committing

The following events are processed in this state:

- [Commit Complete \(section 3.8.7.3\)](#).

3.8.1.2.13 Ended

This is the final state.

The following illustration shows the relationship between the [CONNTYPE_PARTNERTM_BRANCH](#) initiator states.



3.8.1.3 CONNTYPE_PARTNERTM_REDELIVERCOMMIT Acceptor States

The [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) MUST act as an acceptor for the [CONNTYPE PARTNERTM REDELIVERCOMMIT \(section 2.2.9.2.2.1\)](#) connection type. In this role, the subordinate transaction manager MUST provide support for the states in this section.

3.8.1.3.1 Idle

This is the initial state. The following events are processed in this state:

- [Receiving a PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQ message \(section 3.8.5.2.2.1.1\)](#).

3.8.1.3.2 Processing Commit Inquiry

The following events are processed in this state:

- [Commit Complete \(section 3.8.7.3\)](#).

3.8.1.3.3 Ended

This is the final state.

3.8.1.3.4 State Diagram

The following figure shows the relationship between the [CONNTYPE PARTNERTM_REDELIVERCOMMIT](#) acceptor states. Diagram contains two Processing Commit Inquiry boxes but no Ended box.

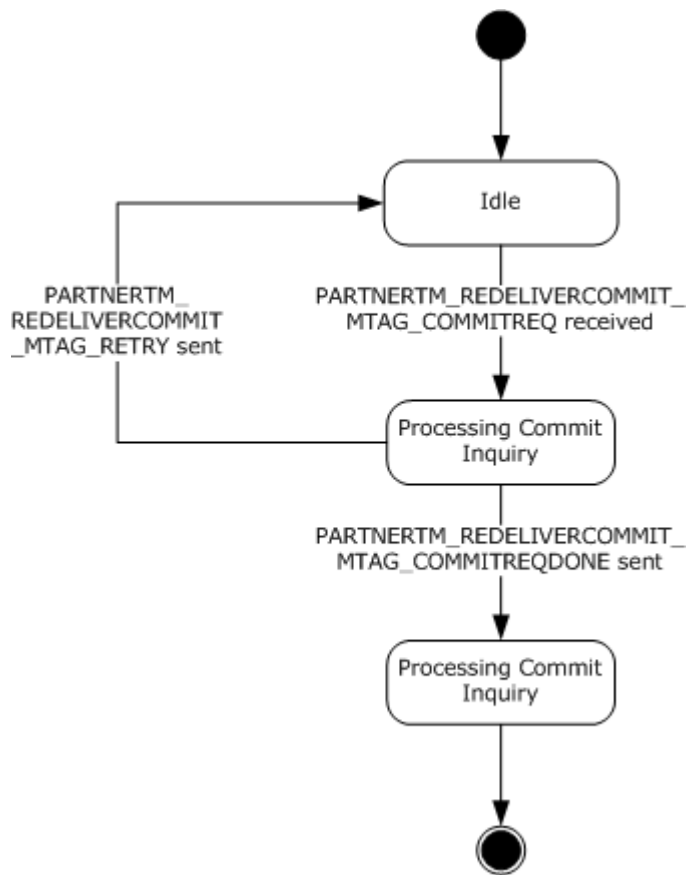


Figure 61: CONNTYPE_PARTNERTM_REDELIVERCOMMIT acceptor states

3.8.1.4 CONNTYPE_PARTNERTM_CHECKABORT Initiator States

The [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) MUST act as an initiator for the [CONNTYPE PARTNERTM CHECKABORT \(section 2.2.9.2.1.1\)](#) connection type. In this role, the subordinate transaction manager MUST provide support for the states in this section.

3.8.1.4.1 Idle

This is the initial state. The following events are processed in this state:

- [Recover In Doubt Transaction \(section 3.8.7.8\)](#).

3.8.1.4.2 Awaiting Confirmation

The following events are processed in this state:

- [Receiving a PARTNERTM_CHECKABORT_MTAG_ABORTED Message \(section 3.8.5.2.1.1.1\)](#).
- [Receiving a PARTNERTM_CHECKABORT_MTAG_RETRY Message \(section 3.8.5.2.1.1.2\)](#).
- [Cancel Check Abort \(section 3.8.7.2\)](#).

3.8.1.4.3 Waiting to Re-Request

The following events are processed in this state:

- [Check Abort Timer \(section 3.8.2.1\)](#).
- [Cancel Check Abort \(section 3.8.7.2\)](#).

3.8.1.4.4 Ended

This is the final state.

3.8.1.4.5 State Diagram

The following figure shows the relationship between the [CONNTYPE PARTNERTM CHECKABORT](#) initiator states.

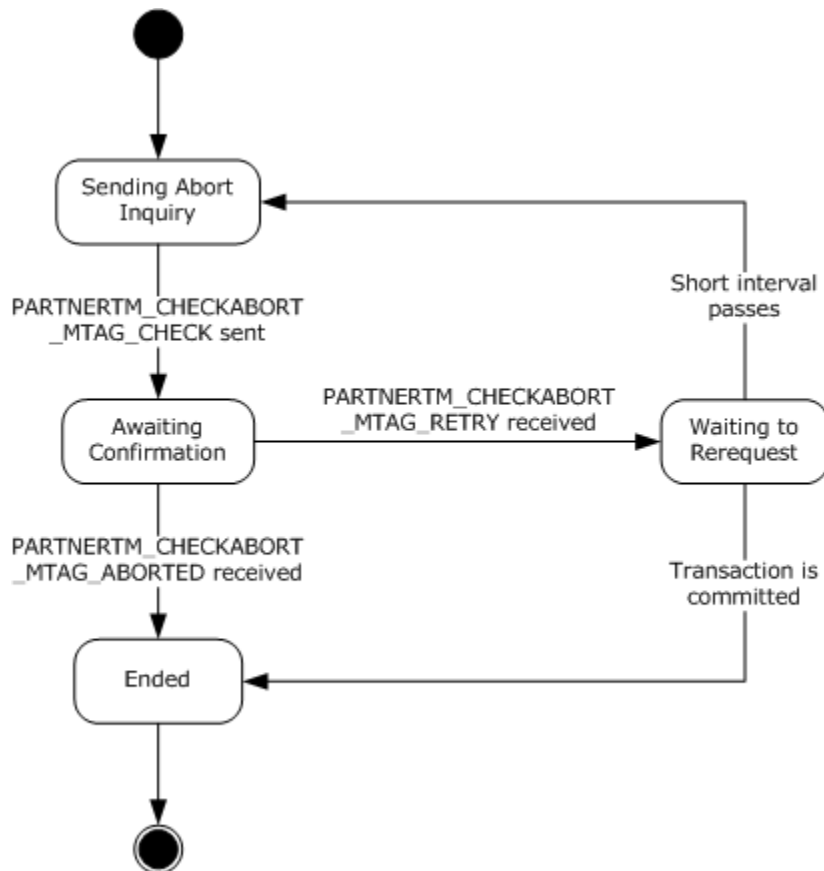


Figure 62: CONNTYPE_PARTNERTM_CHECKABORT initiator states

3.8.2 Timers

The [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST provide the [Check Abort Timer \(section 3.8.2.1\)](#).

3.8.2.1 Check Abort Timer

This timer MUST be set when the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) receives a [PARTNERTM_CHECKABORT_MTAG_RETRY \(section 2.2.9.2.1.1.3\)](#) message on a [CONNTYPE_PARTNERTM_CHECKABORT \(section 2.2.9.2.1.1\)](#) connection. The timer MUST be canceled when the CONNTYPE_PARTNERTM_CHECKABORT (section 2.2.9.2.1.1) connection is disconnected.

The default, minimum, and maximum values of the timer are implementation-specific. [<51>](#)

When the timer is initialized, the Subordinate Transaction Manager Facet (section 1.3.3.3.5) MUST provide an Enlistment object to associate with the timer. When the timer expires, the same Enlistment object MUST be provided alongside the timer notification. The Subordinate Transaction Manager Facet (section 1.3.3.3.5) MUST provide a distinct Reenlist Timeout timer instance for each CONNTYPE_PARTNERTM_CHECKABORT connection.

3.8.3 Initialization

When the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) is initialized:

- The Subordinate Transaction Manager Facet MUST examine the following security flags on the [Core Transaction Manager Facet](#) and perform the following actions:
 - If one of the Allow Network Access, Allow Network Transactions or Allow Inbound Transactions flags is set to false:
 - The Subordinate Transaction Manager Facet MUST refuse to accept incoming connections from remote machines for the following connection types:
 - [CONNTYPE_PARTNERTM_PROPAGATE \(section 2.2.9.1.1.1\)](#)
 - If one of the Allow Network Access or Allow Network Transactions flags is set to false, or if both the Allow Inbound Transactions and Allow Outbound Transactions flags are set to false:
 - The Subordinate Transaction Manager Facet MUST refuse to accept incoming connections from remote machines for the following connection types:
 - [CONNTYPE_PARTNERTM_REDELIVERCOMMIT \(section 2.2.9.2.2.1\)](#)

3.8.4 Higher-Layer Triggered Events

There are no higher-layer triggered events.

3.8.5 Message Processing Events and Sequencing Rules

3.8.5.1 Transaction Propagation and Coordination

3.8.5.1.1 Push Propagation

3.8.5.1.1.1 CONNTYPE_PARTNERTM_PROPAGATE as Acceptor

For all messages received in this connection type, the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) MUST process the message as specified in section [3.1](#).

Also, the Subordinate Transaction Manager Facet (section 1.3.3.3.5) MUST override the default state verification actions for incoming messages as specified in section [3.1.6](#) in the following manner:

- If the current connection state does not define a processing rule for the message:
 - Send a [PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR](#) message using the connection.
 - Perform default invalid message processing, as specified in section [3.1.6](#).

The Subordinate Transaction Manager Facet (section 1.3.3.3.5) MUST additionally follow the processing rules as specified in this section.

3.8.5.1.1.1.1 Receiving a PARTNERTM_PROPAGATE_MTAG_PROPAGATE Message

When the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) receives a [PARTNERTM_PROPAGATE_MTAG_PROPAGATE \(section 2.2.9.1.1.1.1\)](#) message, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Propagating.
 - If the transaction manager does not have sufficient memory available to process the message, it MUST:
 - Send a [PARTNERTM_PROPAGATE_MTAG_NO_MEM \(section 2.2.9.1.1.1.4\)](#) message.
 - Set the connection state to [Ended \(section 3.2.1.4.14\)](#).
 - Otherwise, find the transaction object in the transaction table of the transaction manager using the **guidTx** field from the message as the key:
 - If the transaction object is found in the list:
 - Send a [PARTNERTM_PROPAGATE_MTAG_DUPLICATE \(section 2.2.9.1.1.1.3\)](#) message to the application.
 - Set the connection state to Ended.
 - Otherwise, if the transaction object is not found in the list:
 - Create a new transaction object with the information provided in the message:
 - Use the **guidTx** field from the message as the Identifier value.
 - Use the **isoLevel** field from the message as the Isolation Level value.
 - Use the **isoFlags** field from the message as the Isolation Flags value.
 - Use the **szDesc** field from the message as the Description value.
 - Add the connection to the connection list of the transaction.
 - Create a new enlistment object with the following values:
 - The Subordinate Transaction Manager Facet.
 - The new transaction object.
 - This connection object.
 - Assign the enlistment to the connection's Enlistment field.

- Signal the [Create Superior Enlistment \(section 3.2.7.12\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the enlistment object.
- Otherwise, the message MUST be processed as specified in section [3.1.6](#).

3.8.5.1.1.1.2 Receiving Other PARTNERTM_PROPAGATE_MTAG Messages

When the [Superior Transaction Manager Facet](#) receives one of these messages:

- [PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTRATIONREJECTED \(section 2.2.9.1.1.1.16\)](#)
- [PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTERED \(section 2.2.9.1.1.1.15\)](#)
- [PARTNERTM_PROPAGATE_MTAG_PHASE0 \(section 2.2.9.1.1.1.17\)](#)
- [PARTNERTM_PROPAGATE_MTAG_ABORTREQ \(section 2.2.9.1.1.1.11\)](#)
- [PARTNERTM_PROPAGATE_MTAG_PREPAREREQ \(section 2.2.9.1.1.1.6\)](#)
- [PARTNERTM_PROPAGATE_MTAG_COMMITREQ \(section 2.2.9.1.1.1.9\)](#)

it MUST follow the same message processing rules as the [CONNTYPE PARTNERTM_BRANCH \(section 2.2.9.1.2.1\)](#) connection type acting as an initiator. See section [3.1.6](#) for more information.

3.8.5.1.1.1.3 Receiving a PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR Message

The processing of this event MUST be identical to the processing of the [Connection Disconnected](#) event.

3.8.5.1.1.1.4 CONNTYPE_PARTNERTM_PROPAGATE Connection Disconnected

When a [CONNTYPE PARTNERTM_PROPAGATE \(section 2.2.9.1.1.1\)](#) connection is disconnected, the application MUST perform the same actions as the [CONNTYPE PARTNERTM_BRANCH \(section 2.2.9.1.2.1\)](#) connection type acting as an initiator. See section [3.8.5.1.2.1](#) for more information.

3.8.5.1.2 Pull Propagation

3.8.5.1.2.1 CONNTYPE_PARTNERTM_BRANCH as Initiator

For all messages received in this connection type, the [Subordinate Transaction Manager Facet](#) MUST process the message as specified in section [3.8](#) (and in its subsections).

Also, the Subordinate Transaction Manager Facet MUST override the default state verification actions for incoming messages as specified in section [3.8](#) in the following manner:

- If the current connection state does not define a processing rule for the message:
 - Send a [PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR](#) message using the connection.
 - Perform default invalid message processing, as specified in section [3.8](#) (and in its subsections).

The Subordinate Transaction Manager Facet MUST additionally follow the processing rules as specified in the following sections.

3.8.5.1.2.1.1 Receiving a PARTNERTM_BRANCH_MTAG_BRANCHED Message

When the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) receives a [PARTNERTM_BRANCH_MTAG_BRANCHED \(section 2.2.9.1.2.1.6\)](#) message, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the connection state is Awaiting Branch Response:
 - Set the connection state to Active.
 - Create an Enlistment object with the following values:
 - The Subordinate Transaction Manager Facet.
 - The transaction object referenced by the connection's Connection-Specific Data field.
 - This connection object.
 - Set this connection's **enlistment** field to reference the new Enlistment object.
 - Signal the [Branch Transaction Success \(section 3.2.7.9\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the Enlistment object.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.8.5.1.2.1.2 Receiving a PARTNERTM_BRANCH_MTAG_BRANCH_LOG_FULL, PARTNERTM_BRANCH_MTAG_BRANCH_NO_MEM, PARTNERTM_BRANCH_MTAG_BRANCH_TOO_LATE, PARTNERTM_BRANCH_MTAG_BRANCH_TOO_MANY, or PARTNERTM_BRANCH_MTAG_BRANCH_TX_NOT_FOUND Message

When the [Subordinate Transaction Manager Facet](#) receives one of these messages, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the connection state is Awaiting Branch Response:
 - Create an Enlistment object with a reference to the Subordinate Transaction Manager Facet, a reference to this connection, and a reference to the transaction object referenced by this connection.
 - Signal the [Branch Transaction Failure](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The Enlistment object.
 - The failure code that matches the incoming message:
 - PARTNERTM_BRANCH_MTAG_BRANCH_LOG_FULL: Log Full
 - PARTNERTM_BRANCH_MTAG_BRANCH_NO_MEM: No Mem
 - PARTNERTM_BRANCH_MTAG_BRANCH_TOO_LATE: Too Late
 - PARTNERTM_BRANCH_MTAG_BRANCH_TOO_MANY: Too Many
 - PARTNERTM_BRANCH_MTAG_BRANCH_TX_NOT_FOUND: Tx Not Found
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.1.6](#).

3.8.5.1.2.1.3 Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTERED Message

When the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) receives a [PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTERED \(section 2.2.9.1.1.1.15\)](#) message, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the connection state is Awaiting Registration Response:
 - Set the connection state to Awaiting Phase Zero.
 - Signal the [Register Phase Zero Success \(section 3.2.7.29\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the Enlistment object referenced by this connection.
- Otherwise, if the connection state is Awaiting Registration Response During Phase Zero:
 - Set the connection state to Awaiting Phase Zero Outcome With Outstanding Registration.
 - Signal the Register Phase Zero Success event on the Core Transaction Manager Facet with the Enlistment object referenced by this connection.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.8](#).

3.8.5.1.2.1.4 Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTRATIONREJECTED Message

When the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) receives a [PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTRATIONREJECTED \(section 2.2.9.1.1.1.16\)](#) message, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the connection state is Awaiting Registration Response:
 - Set the connection state to Active.
 - Signal the [Register Phase Zero Failure \(section 3.2.7.28\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the following arguments:
 - The Enlistment object referenced by this connection.
 - A failure reason of Too Late.
- Otherwise, if the connection state is Awaiting Registration Response During Phase Zero:
 - Set the connection state to Awaiting Phase Zero Outcome.
 - Signal the Register Phase Zero Failure event on the Core Transaction Manager Facet with the following arguments:
 - The transaction Enlistment object referenced by this connection.
 - A failure reason of Too Late.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.8](#).

3.8.5.1.2.1.5 Receiving a PARTNERTM_PROPAGATE_MTAG_ABORTREQ Message

When the [Subordinate Transaction Manager Facet](#) receives a [PARTNERTM_PROPAGATE_MTAG_ABORTREQ](#) message, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the connection state is either Active or Prepared:
 - Set the connection state to Aborting.
 - Signal the [Begin Rollback](#) event on the [Core Transaction Manager Facet](#) with the transaction object referenced by the Enlistment object referenced by this connection.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.8](#).

3.8.5.1.2.1.6 Receiving a PARTNERTM_PROPAGATE_MTAG_PHASE0 Message

When the [Subordinate Transaction Manager Facet](#) receives a [PARTNERTM_PROPAGATE_MTAG_PHASE0](#) message, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the connection state is Awaiting Phase Zero:
 - Set the connection state to Awaiting Phase Zero Outcome.
 - Signal the [Begin Phase Zero](#) event on the [Core Transaction Manager Facet](#) with the transaction object referenced by the Enlistment object referenced by this connection.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.8](#).

3.8.5.1.2.1.7 Receiving a PARTNERTM_PROPAGATE_MTAG_PREPAREREQ Message

When the [Subordinate Transaction Manager Facet](#) receives a [PARTNERTM_PROPAGATE_MTAG_PREPAREREQ](#) message, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the connection state is Active:
 - Set the connection state to Preparing.
 - Signal the [Begin Phase One](#) event on the [Core Transaction Manager Facet](#) with the following arguments:
 - The transaction object referenced by the Enlistment object referenced by this connection.
 - The fSinglePhase flag from the message.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.8](#).

3.8.5.1.2.1.8 Receiving a PARTNERTM_PROPAGATE_MTAG_COMMITREQ Message

When the [Subordinate Transaction Manager Facet](#) receives a [PARTNERTM_PROPAGATE_MTAG_COMMITREQ](#) message, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the connection state is Prepared:

- Set the connection state to Committing.
- Signal the [Begin Commit](#) on the [Core Transaction Manager Facet](#) event with the transaction object referenced by the Enlistment object referenced by this connection.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.8](#).

3.8.5.1.2.1.9 Receiving a PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR Message

The processing of this event MUST be identical to the processing of the [Connection Disconnected](#) event.

3.8.5.1.2.1.10 WConnection Disconnected

When a [CONNTYPE PARTNERTM BRANCH \(section 2.2.9.1.2.1\)](#) connection is disconnected, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- If the connection state is Prepared:
 - Signal the [Recover In Doubt Transaction \(section 3.8.7.8\)](#) event on the Subordinate Transaction Manager Facet (section 3.8) with the Enlistment object referenced by this connection.
- Otherwise, if the connection state is Preparing:
 - If the Transaction object's Single Phase Commit flag is false, signal the [Begin Rollback \(section 3.2.7.6\)](#) event on [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the Transaction object referenced by the Enlistment object referenced by this connection.
 - Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).
- Otherwise, if the connection state is Awaiting Branch Response
 - Create an Enlistment object with the following values:
 - The Subordinate Transaction Manager Facet (section 3.8).
 - The Transaction object referenced by the server object referenced by this connection.
 - This Connection object.
 - Signal the [Branch Transaction Failure \(section 3.2.7.8\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the following arguments:
 - The new server object.
 - A failure code of Comm Failed.
 - Set the connection state to Ended.
- Otherwise, if the connection state is Active, Awaiting Registration Response, Awaiting Registration Response During Phase Zero, Awaiting Phase Zero, Awaiting Phase Zero Outcome, or Awaiting Phase Zero Outcome With Outstanding Registration:
 - Signal the [Begin Rollback \(section 3.2.7.6\)](#) event on the Core Transaction Manager Facet (section 1.3.3.3.1) with the Transaction object referenced by this connection.

- Otherwise, the event MUST be processed as specified in section [3.1.8.1](#).

3.8.5.2 Transaction Recovery

3.8.5.2.1 Subordinate-Driven Recovery

3.8.5.2.1.1 CONNTYPE_PARTNERTM_CHECKABORT as Initiator

For all messages received in this connection type, the [Subordinate Transaction Manager Facet](#) MUST process the message in accordance with section [3.8](#). The Subordinate Transaction Manager MUST additionally follow the processing rules specified in the following sections.

3.8.5.2.1.1.1 Receiving a PARTNERTM_CHECKABORT_MTAG_ABORTED Message

When the [Subordinate Transaction Manager Facet](#) (section [1.3.3.3.5](#)) receives a [PARTNERTM_CHECKABORT_MTAG_ABORTED](#) (section [2.2.9.2.1.1.2](#)) message, the Subordinate Transaction Manager MUST perform the following actions:

- If the connection state is Awaiting Confirmation:
 - Signal the [Begin Rollback](#) (section [3.2.7.6](#)) event on the [Core Transaction Manager Facet](#) (section [1.3.3.3.1](#)) with the:
 - Transaction object referenced by the Enlistment object referenced by this connection.
 - Set the Connection state to Ended.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.8](#).

3.8.5.2.1.1.2 Receiving a PARTNERTM_CHECKABORT_MTAG_RETRY Message

When the [Subordinate Transaction Manager Facet](#) receives a [PARTNERTM_CHECKABORT_MTAG_RETRY](#) message, the Subordinate Transaction Manager Facet MUST perform the following actions:

- If the connection state is Awaiting Confirmation:
 - Set the connection state to Idle.
 - Initialize a Check Abort Timeout timer with the following argument:
 - The connection's Enlistment object.
 - An implementation-specific time-out value.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.8](#).

3.8.5.2.1.1.3 CONNTYPE_PARTNERTM_CHECKABORT Connection Disconnected

When a [CONNTYPE_PARTNERTM_CHECKABORT](#) (section [2.2.9.2.1.1](#)) connection is disconnected, the [Subordinate Transaction Manager Facet](#) (section [3.8](#)) MUST perform the following actions:

- If the connection state is Waiting to Rerequest:
 - Cancel the Redeliver Commit Timer associated with the connection.
- If the connection state is Idle, Waiting to Rerequest, or Awaiting Confirmation:

- Signal the [Recover In Doubt Transaction \(section 3.8.7.8\)](#) event on the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) with the Enlistment object referenced by this connection.
- Otherwise, the event MUST be processed as specified in section [3.8](#).

3.8.5.2.2 Superior-Driven Recovery

3.8.5.2.2.1 Receiving a CONNTYPE_PARTNERTM_REDELIVERCOMMIT as Acceptor

For all messages received in this connection type, the [Subordinate Transaction Manager Facet](#) MUST process the message in accordance with section [3.8](#). The Subordinate Transaction Manager Facet MUST additionally follow the processing rules specified in the following sections.

3.8.5.2.2.1.1 Receiving a PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQ Message

When the subordinate transaction manager receives a [PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQ \(section 2.2.9.2.2.1.1\)](#) message, the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) MUST perform the following actions:

- If the connection state is Idle:
 - Set the connection state to Processing Commit Inquiry.
 - Find the transaction object in the transaction manager's transaction table, using the **guidTx** field from the message as a key.
 - If the transaction object is not found:
 - Send a [PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQDONE \(section 2.2.9.2.2.1.2\)](#) message using the connection.
 - Set the connection state to [Ended \(section 3.2.1.4.14\)](#).
 - Otherwise, if the transaction state is either [Phase One Complete \(section 3.2.1.4.8\)](#) or [In Doubt \(section 3.2.1.4.12\)](#):
 - Signal the [Cancel Check Abort \(section 3.8.7.2\)](#) event on the Subordinate Transaction Manager Facet with the transaction object.
 - Signal the [Begin Commit \(section 3.2.7.2\)](#) event on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the transaction object.
 - Otherwise:
 - Send a [PARTNERTM_REDELIVERCOMMIT_MTAG_RETRY \(section 2.2.9.2.2.1.3\)](#) message using the connection.
 - Set the connection state to Idle.
- Otherwise, the message MUST be processed as an invalid message as specified in section [3.8](#).

3.8.5.2.2.1.2 Connection Disconnected

When a [CONNTYPE PARTNERTM REDELIVERCOMMIT \(section 2.2.9.2.2.1\)](#) connection is disconnected, the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) MUST perform the actions as specified in section [3.8](#).

3.8.6 Timer Events

3.8.6.1 Check Abort Timer

When this timer expires, the [Subordinate Transaction Manager Facet \(section 1.3.3.3.5\)](#) MUST perform the following actions:

- If the connection referenced by the provided Enlistment is in the Waiting to Re-request state:
 - Send a [PARTNERTM CHECKABORT MTAG CHECK \(section 2.2.9.2.1.1.1\)](#) message using the connection referenced by the provided Enlistment object:
 - Set the **guidTX** field to the provided Enlistment object's Transaction object's Identifier field.
 - Set the connection state to Awaiting Confirmation.
- Otherwise, ignore the timer event.

3.8.7 Other Local Events

A [Subordinate Transaction Manager Facet](#) MUST be prepared to process the local events defined in the following sections.

The subordinate transaction manager MUST be prepared to process local events pertaining to [Phase Zero](#) functionality only on versions where the connection type [CONNTYPE TXUSER PHASE0](#) is supported. [Connection Types Relevant to Resource Managers - Versioning](#) defines protocol version support for this connection type. The following local events are affected:

- [Register Phase Zero](#)
- [Phase Zero Complete](#)

3.8.7.1 Branch Transaction

The Branch Transaction event MUST be signaled with the following arguments:

- A Transaction object.
- A Name object representing the remote superior transaction manager.

If the Branch Transaction event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- Initiate a new [CONNTYPE PARTNERTM BRANCH \(section 2.2.9.1.2.1\)](#) connection to the provided Name object.
- Assign the provided Transaction object to the connection's Connection-Specific Data field.
- Send a [PARTNERTM BRANCH MTAG BRANCHING \(section 2.2.9.1.2.1.7\)](#) message using the connection:

- Set the **guidTX** field to the provided transaction's identifier.
- Set the connection state to [Awaiting Branch Response \(section 3.8.1.2.2\)](#).

3.8.7.2 Cancel Check Abort

The Cancel Check Abort event MUST be signaled with the following arguments:

- A Transaction object.

If the Cancel Check Abort event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- If a Check Abort timer is active for the transaction, cancel it.
- Find a Connection object of type [CONNTYPE PARTNERTM CHECKABORT \(section 2.2.9.2.1.1\)](#) in the Transaction object's Connection list.
- If no such connection is found, ignore the event.
- Otherwise, set the connection state to Ended.

3.8.7.3 Commit Complete

The Commit Complete event MUST be signaled with the following arguments:

- An Enlistment object.

If the Commit Complete event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- If the provided enlistment's connection is of type [CONNTYPE TXUSER BRANCH \(section 2.2.9.1.2.1\)](#) or [CONNTYPE PARTNERTM PROPAGATE \(section 2.2.9.1.1.1\)](#):
 - Send a [PARTNERTM PROPAGATE MTAG COMMITREQDONE \(section 2.2.9.1.1.1.10\)](#) message using the provided enlistment's connection.
 - Set the connection state to Ended.
- Otherwise, if the provided enlistment's connection is of type [CONNTYPE PARTNERTM REDELIVERCOMMIT \(section 2.2.9.2.2.1\)](#):
 - Send a [PARTNERTM REDELIVERCOMMIT MTAG COMMITREQDONE \(section 2.2.9.2.2.1.2\)](#) message using the provided enlistment's connection.
 - Set the connection state to Ended.

3.8.7.4 Create Superior Enlistment Success

The Create Superior Enlistment Success event MUST be signaled with the following arguments:

- An Enlistment object.

If the Create Superior Enlistment Success event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- Send a [PARTNERTM BRANCH MTAG PROPAGATED \(section 2.2.9.1.1.1.2\)](#) message using the provided enlistment's connection:

- Set the connection state to Active.

3.8.7.5 Create Superior Enlistment Failure

The Create Superior Enlistment event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the failure reason. The reason MUST be set to one of the following values:
 - Duplicate
 - No Mem
 - Log Full

If the Create Superior Enlistment Failure event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- Send the matching message for the following reason codes using the provided enlistment's connection:
 - Duplicate: [PARTNERTM_PROPAGATE_MTAG_DUPLICATE \(section 2.2.9.1.1.1.3\)](#).
 - No Mem: [PARTNERTM_PROPAGATE_MTAG_NO_MEM \(section 2.2.9.1.1.1.4\)](#).
 - Log Full: [PARTNERTM_PROPAGATE_MTAG_LOG_FULL \(section 2.2.9.1.1.1.5\)](#).
- Set the connection state to Ended.

3.8.7.6 Phase Zero Complete

The Phase Zero Complete event MUST be signaled with the following arguments:

- An Enlistment object.
- An outcome value. This value MUST be one of the following:
 - Success
 - Failure

If the Phase Zero Complete event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- If the provided outcome is Success:
 - Send a [PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE \(section 2.2.9.1.1.1.18\)](#) message using the provided enlistment's connection.
 - Set the connection state to Active.
- Otherwise:
 - Send a [PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY \(section 2.2.9.1.1.1.13\)](#) message using the provided enlistment's connection.
 - Set the connection state to Ended.

3.8.7.7 Phase One Complete

The Phase One Complete event MUST be signaled with the following arguments:

- An Enlistment object.
- A value indicating the outcome of [Phase One](#). The value MUST be set to one of the following values:
 - Read Only
 - Prepared
 - Committed
 - Aborted
 - InDoubt

If the Phase One Complete event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- If the provided outcome is Read Only:
 - Send a [PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE \(section 2.2.9.1.1.1.7\)](#) message using the provided enlistment's connection:
 - Set the **prepareReqDone** field to PARTNERTM_PROPAGATE_PREPAREREQDONE_READ_ONLY.
 - Set the connection state to Ended.
- Otherwise, if the provided outcome is Prepared:
 - Send a PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE (section 2.2.9.1.1.1.7) message using the provided enlistment's connection:
 - Set the **prepareReqDone** field to PARTNERTM_PROPAGATE_PREPAREREQDONE_OK.
 - Set the connection state to Ended.
- Otherwise, if the provided outcome is Committed:
 - Send a PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE (section 2.2.9.1.1.1.7) message using the connection:
 - Set the **prepareReqDone** field to PARTNERTM_PROPAGATE_PREPAREREQDONE_SINGLEPHASE_COMMIT.
 - Set the connection state to Ended.
- Otherwise, if the provided outcome is Aborted:
 - Send a PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE (section 2.2.9.1.1.1.7) message using the connection:
 - Set the **prepareReqDone** field MUST to PARTNERTM_PROPAGATE_PREPAREREQDONE_ABORT.

- Set the connection state to Ended.
- Otherwise, if the provided outcome is [In Doubt \(section 3.2.1.4.12\)](#):
 - Send a PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE (section 2.2.9.1.1.1.7) message using the connection:
 - Set the **prepareReqDone** field MUST to PARTNERTM_PROPAGATE_PREPAREREQDONE_SINGLEPHASE_INDOUBT.
 - Set the connection state to Ended.

3.8.7.8 Recover In Doubt Transaction

The Recover In Doubt Transaction event MUST be signaled with the following arguments:

- An Enlistment object.

If the Recover In Doubt Transaction event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- Initiate a new [CONNTYPE PARTNERTM CHECKABORT \(section 2.2.9.2.1.1\)](#) connection using the Name object referenced by the provided Enlistment's Connection field.
- Send a [PARTNERTM CHECKABORT_MTAG_CHECK \(section 2.2.9.2.1.1.1\)](#) message using the connection:
 - Set the **guidTX** field to the provided transaction's identifier.
- Set the connection state to Awaiting Confirmation.

3.8.7.9 Register Phase Zero

The Register Phase Zero event MUST be signaled with the following arguments:

- An Enlistment object.

If the Register Phase Zero event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- If the provided enlistment's connection state is Active:
 - Set the connection state to [Awaiting Registration Response \(section 3.8.1.1.5\)](#).
 - Send a [PARTNERTM PROPAGATE_MTAG_PHASE0REGISTER \(section 2.2.9.1.1.1.14\)](#) message using the connection.
- Otherwise, if the connection state is Awaiting Phase Zero Outcome:
 - Set the connection state to [Awaiting Registration Response During Phase Zero \(section 3.8.1.1.8\)](#).
 - Send a PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER (section 2.2.9.1.1.1.14) message using the Enlistment's Connection.
- Otherwise:

- Signal the [Register Phase Zero Failure \(section 3.2.7.28\)](#) on the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) with the following arguments:
 - The provided Enlistment object.
 - The Too Late reason code.

3.8.7.10 Rollback Complete

The Rollback Complete event MUST be signaled with the following arguments:

- An Enlistment object.

If the Rollback Complete event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- If the provided enlistment's connection is of type [CONNTYPE PARTNERTM BRANCH \(section 2.2.9.1.2.1\)](#) or [CONNTYPE PARTNERTM PROPAGATE \(section 2.2.9.1.1.1\)](#)
 - Send a [PARTNERTM PROPAGATE MTAG ABORTREQDONE \(section 2.2.9.1.1.1.12\)](#) message using the provided enlistment's connection.
- Otherwise, ignore the signal.

3.8.7.11 Unilaterally Aborted

The Unilaterally Aborted event MUST be signaled with the following arguments:

- An Enlistment object.

If the Unilaterally Aborted event is signaled, the [Subordinate Transaction Manager Facet \(section 3.8\)](#) MUST perform the following actions:

- If the provided enlistment's connection state is Aborting:
 - Ignore the signal.
- Otherwise:
 - Send a [PARTNERTM PROPAGATE MTAG ABORTNOTIFY \(section 2.2.9.1.1.1.13\)](#) message using the provided enlistment's connection.
 - Set the connection state to Ended.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the MSDTC Connection Manager: OleTx Transaction Protocol. These protocol examples generally assume that an OleTX transports session, as specified in [\[MS-CMPO\]](#) section 3.2.1.2, has already been established between the two participants. However, some examples exhibit how one participant establishes a new OleTX transports session with another participant because of the protocol that is being demonstrated.

Participants communicate with each other by using OleTX multiplexing connections, as specified in [\[MS-CMP\]](#) section 3.1.1.1, that are in turn layered on top of the OleTX transports infrastructure (as specified in [\[MS-CMPO\]](#) section 3.2.1). In these examples, messages are sent from one participant to another by submitting a [MESSAGE PACKET \(section 2.2.4.1\)](#) to the underlying OleTX multiplexing layer, as specified in [\[MS-CMP\]](#) section 3.1.4.1.

4.1 Simple Transaction Scenario

This scenario shows how an application creates and completes a transaction. The scenario begins with the application establishing a transport session with a transaction manager and negotiating its connection resources.

4.1.1 Beginning a Transaction

This packet sequence is initiated by starting a connection on a transport session between an application and a transaction manager.

[CONNTYPE_TXUSER_BEGIN2](#): The packet sequence starts when an application initiates a connection using CONNTYPE_TXUSER_BEGIN2.

Field	Value	Value description
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00000028	CONNTYPE_TXUSER_BEGIN2
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The application then sends a [TXUSER_BEGIN2 MTAG_BEGIN](#) user message specifying the isolation level, time-out, transaction description, and isolation flag. For this example, the application requests a transaction with ISOLATIONLEVEL_SERIALIZABLE, a time-out of 60 seconds, a description of "sample transaction", and ISOFLAG_RETAIN_DONTCARE.

Field	Value	Value description
MsgTag	0x000000FF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1

Field	Value	Value description
dwUserMsgType	0x00000002	TXUSER_BEGIN2_MTAG_BEGIN
dwcbVarLenData	0x00000034	52
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
isoLevel	0x00100000	ISOLATIONLEVEL_SERIALIZABLE
dwTimeout	0x0000EA60	60000
szDesc	0x706D6173 0x7420656C 0x736E6172 0x69746361 0x00006E6F 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000	"sample transaction"
isoFlags	0x00000005	ISOFLAG_RETAIN_DONTCARE

When the transaction manager receives the TXUSER_BEGIN2_MTAG_BEGIN message from the application, the transaction manager attempts to create a transaction object with a globally unique identifier (GUID) as its identifier. If the transaction manager successfully creates the transaction, it sends a [TXUSER_BEGIN2_MTAG_SINK_BEGUN](#) user message to the application specifying the transaction identifier as the guidTx field (for example, 4046037e-9722-46c9-9883-99062341cb35), and the transaction manager adds the transaction to its list of known transaction objects.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00000028	TXUSER_BEGIN2_MTAG_SINK_BEGUN
dwcbVarLenData	0x00000010	16
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidTx	0x4046037E 0x46C99722 0x06999883 0x35CB4123	4046037e-9722-46c9-9883-99062341cb35

4.1.2 Completing a Transaction

After the transaction begins, the application decides whether to commit or abort the transaction. If the application disconnects the connection before committing or aborting the transaction, the transaction manager assumes that the transaction aborts.

4.1.2.1 Committing the Transaction

The application commits the transaction by sending a [TXUSER_BEGIN2_MTAG_COMMIT](#) user message specifying a value of zero in the unused **grfRM** field.

Field	Value	Value description
MsgTag	0x000000FF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00006003	TXUSER_BEGIN2_MTAG_COMMIT
dwcbVarLenData	0x00000004	4
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
grfRM	0x00000000	0

In response, the transaction manager attempts to commit the transaction by using a two-phase commit. If the transaction manager successfully completes [Phase One](#) of the transaction, the transaction manager sends a [TXUSER_BEGIN2_MTAG_SINK_ERROR](#) user message to the application with TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED specified in the **Error** field.

Field	Value	Value description
MsgTag	0x000000FF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00006005	TXUSER_BEGIN2_MTAG_SINK_ERROR
dwcbVarLenData	0x00000004	4
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
Error	0x0000001F	TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED

After the application gets the TXUSER_BEGIN2_MTAG_SINK_ERROR response from its transaction manager, no more user messages can be sent on this connection and the application initiates the disconnect sequence.

4.2 Transaction Marshaling Scenario (Pull Propagation)

This scenario shows how an application (or resource manager) on Machine1 marshals an existing transaction to an application or resource manager on Machine2 by using pull propagation. Because the receiving application obtains knowledge of an existing transaction, it is implied that another

application exists that has access to an existing transaction from which the receiving application will obtain the necessary information. Because OleTx does not prescribe application-to-application communication, an out-of-band mechanism (such as an application API) needs to be available by which this knowledge is transferred from the sending application to the receiving application.

Pull propagation involves three main stages. In the first stage, the sending application (or sender) packages information about an existing transaction and sends the information to the receiving application (or receiver) — this is called marshaling the transaction.

During the second stage (unmarshaling the transaction), the receiver requests an association with the transmitted transaction. If the transaction manager of the receiver does not have a reference for the requested transaction, it enters the third stage and attempts to add itself as a subordinate branch of the transaction using the transaction manager of the sender.

This scenario requires that the receiving application has established a transport session with a transaction manager and has negotiated its connection resources. The scenario also assumes that there is an out-of-band mechanism (an application API) that the sending and receiving applications use to exchange transactional information. In general, this API is also necessary for the sending application to prescribe work for the receiving application to perform as part of the transaction.

4.2.1 Marshaling the Transaction

To marshal a transaction from the sending application to the receiving application, several pieces of information need to be transmitted to the receiver. The receiver needs to have sufficient knowledge of the existing transaction. That knowledge includes the transaction identifier, the isolation level, the isolation flag, and the description of the transaction.

The receiver also needs to have sufficient locative information of the sender's transaction manager in order for the receiver's transaction manager to establish a communication session with the sender's transaction manager (that is, the [OLETX_TM_ADDR](#)). The OLETX_TM_ADDR includes the sender's transaction manager's host name, its contact identifier, and the RPC communication protocols that the sender's transaction manager supports.

Field	Value	Description
dwVersionMin	0x00000001	1
dwVersionMax	0x00000002	2
guidTx	0x4046037E 0x46C99722 0x06999883 0x35CB4123	4046037e-9722-46c9-9883-99062341cb35
isoLevel	0x00100000	ISOLATIONLEVEL_SERIALIZABLE
isoFlags	0x00000006	ISOFLAG_RETAIN_DONTCARE
cbSourceTmAddr	0x00000058	88
szDesc	0x706D6173 0x7420656C 0x736E6172 0x69746361 0x00006E6F	"sample-transaction"

Field	Value	Description
	0x00000000 0x00000000 0x00000000 0x00000000 0x00000000	
szGuid	0x36343034 0x65373330 0x3237392D 0x36342D32 0x392D3963 0x2D333838 0x36303939 0x31343332 0x35336263 0x00000000	"4046037e-9722-46c9-9883-99062341cb35"
dwcbHostName	0x0000000a	10
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
grbComProtsSupported	0x00000021	PROT_IP_TCP PROT_LRPC
szHostName	0x6863614d 0x5f656e69 0x00000031	"Machine_1"
cbHostNameW	0x00000014	20
wszHostname	0x0061004D 0x00680063 0x006E0069 0x005F0065 0x00000031	L"Machine_1"

4.2.2 Unmarshaling the Transaction

To begin the unmarshaling process, the receiving application initiates a connection over its transport session with its transaction manager.

[CONNTYPE_TXUSER_ASSOCIATE](#): The packet sequence starts when the receiving application initiates a connection by using CONNTYPE_TXUSER_ASSOCIATE.

Field	Value	Value description
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1

Field	Value	Value description
dwUserMsgType	0x00000011	CONNTYPE_TXUSER_ASSOCIATE
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The receiving application then sends a [TXUSER_ASSOCIATE MTAG_ASSOCIATE](#) user message with the information transmitted to the receiver in the [Propagation Token](#).

Field	Value	Value description
MsgTag	0x000000FF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002031	TXUSER_ASSOCIATE_MTAG_ASSOCIATE
dwcbVarLenData	0x0000007C	124
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidTx	0x4046037E 0x46C99722 0x06999883 0x35CB4123	4046037e-9722-46c9-9883-99062341cb35
isoLevel	0x00100000	ISOLATIONLEVEL_SERIALIZABLE
isoFlags	0x00000005	ISOFLAG_RETAIN_DONTCARE
cbSourceTmAddr	0x00000038	56
szDesc	0x706D6173 0x7420656C 0x736E6172 0x69746361 0x00006E6F 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000	"sample transaction"
guidSignature	0xBAA04775 0x4F498F43 0x1B5AADEF 0x0B195121	DC85CB48-D8A5-11d2-828B-00805F0DF75A
guidEndpoint	0xBAA04775 0x4F498F43	BAA04775-8F43-4F49-ADEF-5A1B2151190B

Field	Value	Value description
	0x1B5AADEF 0x0B195121	
grbComProtsSupported	0x00000021	PROT_IP_TCP PROT_LRPC
wszHostName	0x0061004d 0x00680063 0x006e0069 0x005f0065 0x00000031	L"Machine_1"

When the receiver's transaction manager receives the TXUSER_ASSOCIATE_MTAG_ASSOCIATE message, that transaction manager attempts to locate the transaction in its list of transaction objects by using the transaction identifier. If the transaction object is not found locally, the transaction manager attempts to pull the transaction from the sender's transaction manager by using information contained in the Propagation Token (compare [Branching the Transaction](#)).

If the receiver's transaction manager can successfully locate the transaction object or if the requested transaction is successfully pulled to the receiver's transaction manager, it replies to the receiver with a [TXUSER_ASSOCIATE_MTAG_ASSOCIATED](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002032	TXUSER_ASSOCIATE_MTAG_ASSOCIATED
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

After the receiving application gets the TXUSER_ASSOCIATE_MTAG_ASSOCIATED response from its transaction manager (or if it receives an error response), no more user messages can be sent on this connection and the receiver initiates the disconnect sequence.

4.2.3 Branching the Transaction

If the receiver's transaction manager does not have a reference to the requested transaction in its list of transaction objects, it attempts to contact the sender's transaction manager. If successful, it requests a subordinate branch to the transaction through the sender's transaction manager.

To branch the transaction, the receiver's transaction manager needs to have a transport session with the sender's transaction manager. If there is no existing transport session, the receiver's transaction manager uses the OLETX_TM_ADDR information about the sender's transaction manager from the [Propagation Token \(section 2.2.5.4\)](#) to initiate a session between the two participants. Depending on the value of both participants' contact identifiers, the receiver's transaction manager initiates the transport session as either the primary or secondary partner.

To branch the transaction, the receiver's transaction manager initiates a connection over its transport session with the sender's transaction manager. If the transaction branching is successful, the superior transaction manager (that is, the sender's transaction manager) adds the receiver's transaction manager as a subordinate branch to the transaction.

[CONNTYPE_PARTNERTM_BRANCH](#): The packet sequence starts when the receiver's transaction manager initiates a CONNTYPE_PARTNERTM_BRANCH connection with the sender's transaction manager.

Field	Value	Value description
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00000104	CONNTYPE_PARTNERTM_BRANCH
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The receiver's transaction manager then sends a [PARTNERTM_BRANCH_MTAG_BRANCHING](#) user message with the transaction identifier of the requested transaction.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002051	PARTNERTM_BRANCH_MTAG_BRANCHING
dwcbVarLenData	0x00000010	16
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidTx	0x4046037E 0x46C99722 0x06999883 0x35CB4123	4046037e-9722-46c9-9883-99062341cb35

If the sender's transaction manager is able to create a subordinate branch, it responds to the receiver's transaction manager with a user message with dwUserMsgType equal to [PARTNERTM_BRANCH_MTAG_BRANCHED](#).

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1

Field	Value	Value description
dwUserMsgType	0x00002052	PARTNERTM_BRANCH_MTAG_BRANCHED
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

After receiving the PARTNERTM_BRANCH_MTAG_BRANCHED reply from the sender's transaction manager, the receiver's transaction manager keeps the connection open in order to process two-phase commit notifications from the sender's transaction manager. The sender's transaction manager has now become the superior transaction manager for this transaction; the receiver's transaction manager is now the subordinate transaction manager.

If the sender's transaction manager is unable to create a subordinate branch, it responds to the receiver's transaction manager with a user message with dwUserMsgType set to an error value. No more messages are sent on this connection and the receiver's transaction manager initiates the disconnect sequence. The receiver transaction manager then sends an appropriate error response to the receiver on the [TXUSER ASSOCIATE MTAG ASSOCIATE](#) connection to inform the receiver of the failure to pull the transaction.

4.3 Transaction Marshaling Scenario (Push Propagation)

This scenario shows how an application or resource manager obtains access to an existing transaction through its transaction manager by using push propagation. Because the receiving application obtains knowledge of an existing transaction, this knowledge implies that there is another application that has access to an existing transaction from which the receiving application will obtain the necessary information. Because OleTx does not prescribe application-to-application communication, there needs to be an out-of-band mechanism (such as an application API) by which this knowledge is exchanged between the sending application and the receiving application.

Push propagation involves four main exchanges. The push sequence begins by the sending application (or sender) obtaining location information from the receiving application (or receiver) about its transaction manager, which is called the whereabouts. Subsequently, the sender uses the receiver's transaction manager whereabouts information to export the transaction. This exchange causes the sender's transaction manager to propagate the transaction to the receiver's transaction manager. The exchanges complete when the receiver imports the transaction from its transaction manager.

The scenario requires that the receiving application has established a transport session with a transaction manager and has negotiated its connection resources. The scenario also assumes that there is some out-of-band mechanism (an application API) that the sending and receiving applications use to exchange transactional information. In general, this API will also be necessary for the sending application to prescribe work for the receiving application to perform as part of the transaction.

4.3.1 Obtaining the Whereabouts of the Receiver's Transaction Manager

To push the transaction from the sender's transaction manager to the receiver's transaction manager, the sender obtains the location of the receiver's transaction manager. Specifically, the sender needs to populate an [OLETX_TM_ADDR](#) structure in order to perform an export. Typically, the receiver sends an [SWhereabouts](#) structure to the sender by using an out-of-band API. The SWhereabouts structure in this example contains two [STmToTmProtocol](#) structures: [SDtcCmEndpointInfoV1](#) and [SDtcCmEndpointInfoV2](#).

Field	Value	Value description
guidSignature	0x2ADB4462 0x11D0BD41 0xC000B12E 0xEFF3C24F	2ADB4462-BD41-11D0-B12E-00C04FC2F3EF
cTmToTmProtocols	0x00000002	2
tmprotDescribed	0x00000002	TmProtocolMsdtcV1
cbTmProtocolData	0x0000001C	28
comprotSupported	0x00000021	PROT_IP_TCP PROT_LRPC
guidEndpointID	0xD2A6A4B9 0x48ABCDB0 0x34E3A68F 0x28611A9B	D2A6A4B9-CDB0-48AB-A68F-E3349B1A6128
szHostname	0x6863614d 0x00026e69	"Machine_2"
tmprotDescribed	0x00000003	TmProtocolMsdtcV2
cbTmProtocolData	0x00000014	20
wszHostname	0x0061004d 0x00680063 0x006e0069 0x005f0065 0x00000031	L"Machine_2"

4.3.2 Exporting the Transaction

To export the transaction, the sending application needs to have a [CONNTYPE_TXUSER_EXPORT](#) connection established with the transaction manager. If a connection is not established, the sender needs to initiate one now.

CONNTYPE_TXUSER_EXPORT: The packet sequence starts when the sender initiates a CONNTYPE_TXUSER_EXPORT connection with its transaction manager.

Field	Value	Value description
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000002	2
dwUserMsgType	0x00000004	CONNTYPE_TXUSER_EXPORT
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The sending application then sends to its transaction manager a [TXUSER_EXPORT_MTAG_CREATE](#) user message on that connection specifying the receiver's transaction manager in an [OLETX_TM_ADDR](#) structure.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001041	TXUSER_EXPORT_MTAG_CREATE
dwcbVarLenData	0x00000038	56
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidSignature	0xBAA04775 0x4F498F43 0x1B5AADEF 0x0B195121	DC85CB48-D8A5-11d2-828B-00805F0DF75A
guidEndpoint	0xD2A6A4B9 0x48ABCDB0 0x34E3A68F 0x28611A9B	D2A6A4B9-CDB0-48AB-A68F-E3349B1A6128
grbComProtsSupported	0x00000021	PROT_IP_TCP PROT_LRPC
wszHostName	0x0061004d 0x00680063 0x006e0069 0x005f0065 0x00000032	L"Machine_2"

When the sender's transaction manager receives the create message, it sets up a session with the receiver's transaction manager if a session does not already exist. If the connection is successfully created, the transaction manager responds to the sender with a [TXUSER_EXPORT_MTAG_CREATED](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001042	TXUSER_EXPORT_MTAG_CREATED
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

After the export connection is created, the sender requests that the transaction be exported to the receiver's transaction manager by sending a [TXUSER_EXPORT_MTAG_EXPORT](#) user message to its transaction manager specifying the identifier of the transaction the sender wants to have exported in the **guidTx** field of the message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001043	TXUSER_EXPORT_MTAG_EXPORT
dwcbVarLenData	0x00000010	16
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidTx	0x4046037E 0x46C99722 0x06999883 0x35CB4123	4046037e-9722-46c9-9883-99062341cb35

When the sender's transaction manager receives the export message, it attempts to propagate the transaction to the receiver's transaction manager. If the propagation is successful, the transaction manager sends to the sender a [TXUSER_EXPORT_MTAG_EXPORTED](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001044	TXUSER_EXPORT_MTAG_EXPORTED
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

When the sender receives the exported message, it sends information to the receiving application by using an out-of-band API that the exported transaction can be imported.

The sender can either close the export connection with its transaction manager by initiating the disconnect sequence, or it can maintain the connection for future exporting of transactions to the receiver's transaction manager.

4.3.3 Propagating the Transaction

When the sending transaction manager receives the export message from the sending application, the transaction manager attempts to propagate the transaction to the receiving transaction manager. If a transport session has not yet been established, the sending transaction manager attempts to establish the session now.

After a transport session is established between the sending transaction manager and the receiving transaction manager and resources are negotiated, the sending transaction manager initiates a [CONNTYPE_PARTNERTM_PROPAGATE](#) connection with the receiving transaction manager.

CONNTYPE_PARTNERTM_PROPAGATE: The packet sequence starts when the sending transaction manager initiates a CONNTYPE_PARTNERTM_PROPAGATE connection with the receiving transaction manager.

Field	Value	Value description
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00000101	CONNTYPE_PARTNERTM_PROPAGATE
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The sending transaction manager then sends a [PARTNERTM_PROPAGATE_MTAG_PROPAGATE](#) user message to the receiving transaction manager and specifies the transaction identifier (guidTx), the isolation level (isoLevel), the transaction description (szDesc), and the isolation flags (isoFlags).

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002001	PARTNERTM_PROPAGATE_MTAG_PROPAGATE
dwcbVarLenData	0x00000040	64
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidTx	0x4046037E 0x46C99722 0x06999883 0x35CB4123	4046037e-9722-46c9-9883-99062341cb35
isoLevel	0x00100000	ISOLATIONLEVEL_SERIALIZABLE
szDesc	0x706D6173 0x7420656C 0x736E6172 0x69746361 0x00006E6F 0x00000000 0x00000000 0x00000000	"sample transaction"

Field	Value	Value description
	0x00000000 0x00000000	
isoFlags	0x00000005	ISOFLAG_RETAIN_DONTCARE

When the receiving transaction manager receives the message, it adds the transaction to its list of known transactions. If the propagation is successful, the receiving transaction manager sends to the sending transaction manager a [PARTNERTM_PROPAGATE_MTAG_PROPAGATED](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000002	2
dwUserMsgType	0x00002002	PARTNERTM_PROPAGATE_MTAG_PROPAGATED
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

When the sending transaction manager receives the PARTNERTM_PROPAGATE_MTAG_PROPAGATED message, it adds the receiving transaction manager as a subordinate branch to its list of enlistments for the transaction. If the subordinate transaction manager is successfully added, the sending transaction manager replies to the sender that the export completed successfully.

The sending transaction manager keeps the connection alive for future two-phase commit processing when the transaction is committed or aborted.

4.3.4 Importing the Transaction

When the sender receives notification that the transaction was successfully exported to the receiving transaction manager, the sender sends the transaction identifier (guidTx) to the receiver by using its out-of-band API so that the receiver can import the transaction.

To import the transaction, the receiver needs to initiate a [CONNTYPE_TXUSER_IMPORT](#) connection with its transaction manager.

CONNTYPE_TXUSER_IMPORT: The packet sequence starts when the receiver initiates a CONNTYPE_TXUSER_IMPORT connection with its transaction manager.

Field	Value	Value Description
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00000002	CONNTYPE_TXUSER_IMPORT
dwcbVarLenData	0x00000000	0

Field	Value	Value Description
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The receiver then sends a [TXUSER_IMPORT_MTAG_IMPORT](#) user message to the its transaction manager and specifies the transaction identifier (guidTx).

Field	Value	Value Description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00001021	TXUSER_IMPORT_MTAG_IMPORT
dwcbVarLenData	0x00000010	16
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidTx	0x4046037E 0x46C99722 0x06999883 0x35CB4123	4046037e-9722-46c9-9883-99062341cb35

When the transaction manager receives the TXUSER_IMPORT_MTAG_IMPORT message from the receiver, it attempts to find the transaction identifier (guidTx) in its list of known transactions. If it locates the transaction with that identifier, the transaction manager replies to the receiver with a [TXUSER_IMPORT_MTAG_IMPORTED](#) user message that specifies the isolation level (isoLevel) and isolation flags (isoFlags) of the transaction.

Field	Value	Value Description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00001022	TXUSER_IMPORT_MTAG_IMPORTED
dwcbVarLenData	0x00000008	8
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
isoLevel	0x00100000	ISOLATIONLEVEL_SERIALIZABLE
isoFlags	0x00000005	ISOFLAG_RETAIN_DONTCARE

When the receiving application gets the TXUSER_IMPORT_MTAG_IMPORTED message, it can enlist on the transaction (if it is a resource manager) or marshal the transaction to another application. The receiving application can also attempt to abort the transaction by using the connection. If the receiver does not intend to abort the transaction, it initiates the disconnect sequence.

4.4 Simple Enlistment Scenario

This scenario shows how a resource manager registers with a transaction manager, enlists on an existing transaction, and then responds to the enlistment notifications from the transaction manager. This scenario does not address resource manager recovery, which is described in the next section.

The scenario begins by the resource manager establishing a transport session with a transaction manager and negotiating its connection resources. It also assumes that there is some out-of-band mechanism (for example, application API) by which an external application is able to send the resource manager work to perform as part of an existing transaction. The resource manager is expected to follow the two-phase commit protocol.

4.4.1 Registering with the Transaction Manager as a Resource Manager

Before a resource manager can participate in transactional work, it needs to register as a resource manager with a transaction manager.

[CONNTYPE_TXUSER_RESOURCEMANAGER](#): The packet sequence starts when the resource manager initiates a CONNTYPE_TXUSER_RESOURCEMANAGER connection.

Field	Value	Value description
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000002	2
dwUserMsgType	0x00000005	CONNTYPE_TXUSER_RESOURCEMANAGER
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The resource manager then sends a [TXUSER_RESOURCEMANAGER_MTAG_CREATE](#) user message that specifies a GUID that uniquely identifies the resource manager (guidRm) and a second GUID that uniquely identifies this session of the resource manager (guidSession). The session GUID can be either a unique GUID that is created each time the resource manager starts up, or NULL GUID.

Field	Value	Value description
MsgTag	0x000000FF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00001051	TXUSER_RESOURCEMANAGER_MTAG_CREATE
dwcbVarLenData	0x00000020	32
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidRm	0xE7BAEBDF 0x4E2BDC69	E7BAEBDF-DC69-4E2B-9FF1-69A1D3592877

Field	Value	Value description
	0xA1699FF1 0x772859D3	
guidSession	0x8F5204B3 0x466A5FB9 0xAF2DA0B8 0xAAD9CB3F	8F5204B3-5FB9-466A-A0B8-2DAF3FCBD9AA

If guidRm does not identify a resource manager already registered with the transaction manager, the transaction manager adds the resource manager to its list of registered resource managers and sends to the resource manager a [TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00001053	TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The resource manager needs to keep this connection open for the duration of its lifetime. If the connection is terminated, any unprepared transactions are aborted.

4.4.2 Enlisting in an Existing Transaction

To enlist in an existing transaction, the resource manager needs to have knowledge of the existing transaction, which likely happened as a result of marshaling the transaction from an application to the resource manager.

[CONNTYPE_TXUSER_ENLISTMENT](#): The packet sequence starts when the resource manager initiates a connection by using CONNTYPE_TXUSER_ENLISTMENT.

Field	Value	Value description
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000002	2
dwUserMsgType	0x00000003	TXUSER_ENLISTMENT_MTAG_ENLIST
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The resource manager then sends a [TXUSER_ENLISTMENT_MTAG_ENLIST](#) user message specifying the transaction identifier (guidTx), the resource manager identifier (guidRm), and the resource manager session identifier (guidSession).

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001031	TXUSER_ENLISTMENT_MTAG_ENLIST
dwcbVarLenData	0x00000030	48
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidTx	0x4046037E 0x46C99722 0x06999883 0x35CB4123	4046037e-9722-46c9-9883-99062341cb35
guidRm	0xE7BAEBDF 0x4E2BDC69 0xA1699FF1 0x772859D3	E7BAEBDF-DC69-4E2B-9FF1-69A1D3592877
guidSession	0x8F5204B3 0x466A5FB9 0xAF2DA0B8 0xAAD9CB3F	8F5204B3-5FB9-466A-A0B8-2DAF3FCBD9AA

If the transaction manager can enlist the resource manager in the requested transaction, the transaction manager adds the resource manager to its list of subordinate enlistments and replies to the resource manager with a [TXUSER_ENLISTMENT_MTAG_ENLISTED](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001032	TXUSER_ENLISTMENT_MTAG_ENLISTED
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The resource manager is now enlisted on the transaction and is now waiting for two-phase commit notifications from the transaction manager. During the time that the resource manager is enlisted on the transaction, the resource manager typically receives from some external application the instructions (that is, work) to perform as part of the transaction.

4.4.3 Responding to Enlistment Notifications

When the transaction is committed, the transaction manager receives notification to prepare the transaction.

4.4.3.1 Responding to a Prepare Request Message

As part of the prepare process, the transaction manager sends [TXUSER_ENLISTMENT_MTAG_PREPAREREQ](#) user messages to each of its subordinate resource managers.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001033	TXUSER_ENLISTMENT_MTAG_PREPAREREQ
dwcbVarLenData	0x00000008	8
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
grfRM	0x00000000	0
fSinglePhase	0x00000000	0

When the resource manager successfully completes its prepare work, it replies to its transaction manager by using a [TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE](#) user message that has the prepareReqDone value set to TXUSER_ENLISTMENT_PREPAREREQDONE_OK.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001036	TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE
dwcbVarLenData	0x00000014	20
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
prepareReqDone	0x00000000	TXUSER_ENLISTMENT_PREPAREREQDONE_OK
guidReason	0x00000000 0x00000000 0x00000000 0x00000000	00000000-0000-0000-0000-000000000000

The resource manager now waits for the transaction outcome from its transaction manager.

4.4.3.2 Responding to a Commit Request Message

If the transaction manager receives notification that the transaction is committed, it sends to the resource manager a [TXUSER_ENLISTMENT_MTAG_COMMITREQ](#) message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00001035	TXUSER_ENLISTMENT_MTAG_COMMITREQ
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

When the resource manager successfully completes its commit work, it replies to its transaction manager with a [TXUSER_ENLISTMENT_MTAG_COMMITREQDONE](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00001038	TXUSER_ENLISTMENT_MTAG_COMMITREQDONE
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The resource manager has now completed all its work that is associated with the transaction and initiates the disconnect sequence on its [CONNTYPE_TXUSER_ENLISTMENT](#) connection with its transaction manager.

4.5 Transaction Manager Two-Phase Commit Scenario

This scenario shows how a transaction manager performs the Two-Phase Commit Protocol, as both the [superior transaction manager facet](#) and the [subordinate transaction manager facet](#).

For this scenario, all connections that are associated with the transaction are extant. The root transaction manager has an existing [CONNTYPE_TXUSER_BEGIN2](#) connection between itself and the initiating application. Optionally, the transaction has one or more existing [CONNTYPE_PARTNERTM_BRANCH](#) or [CONNTYPE_PARTNERTM_PROPAGATE](#) connections between a superior transaction manager facet and its subordinate transaction manager facets. (A subordinate transaction manager facet can also act as a superior transaction manager facet, if it has any subordinate branches.) Optionally, each transaction manager also has one or more [CONNTYPE_TXUSER_ENLISTMENT](#) connections with its registered resource managers.

For this scenario, it is assumed that there are no phase-zero or voter enlistments and that the root transaction manager has more than one subordinate branch and thus will not perform single-phase commit.

4.5.1 Phase One

The protocol sequence begins when the root transaction manager receives the [TXUSER_BEGIN2_MTAG_COMMIT](#) user message from the initiating application over its existing [CONNTYPE_TXUSER_BEGIN2](#) connection (compare [Committing the Transaction](#)).

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00006003	TXUSER_BEGIN2_MTAG_COMMIT
dwcbVarLenData	0x00000004	4
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
grfRM	0x00000000	0

The root transaction manager then iterates through the subordinate branches of each transaction and notifies the subordinates that the transaction processing has begun. The root transaction manager then waits for reply notifications from each of the subordinates in order to determine the outcome of the transaction.

4.5.1.1 Phase One - Subordinate Resource Managers

If the subordinate branch is a resource manager (that is, using a [CONNTYPE_TXUSER_ENLISTMENT](#) connection), the transaction manager sends a [TXUSER_ENLISTMENT_MTAG_PREPAREREQ](#) user message with fSinglePhase set to zero.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001033	TXUSER_ENLISTMENT_MTAG_PREPAREREQ
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
grfRM	0x00000000	0
fSinglePhase	0x00000000	0

When the resource manager successfully completes its preparation work, it replies to its transaction manager by using a [TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE](#) user message that has the prepareReqDone value set to TXUSER_ENLISTMENT_PREPAREREQDONE_OK.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001036	TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE
dwcbVarLenData	0x00000014	20
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
prepareReqDone	0x00000000	TXUSER_ENLISTMENT_PREPAREREQDONE_OK
guidReason	0x00000000 0x00000000 0x00000000 0x00000000	00000000-0000-0000-0000-000000000000

The resource manager now waits for a [TXUSER_ENLISTMENT_MTAG_ABORTREQ](#) or [TXUSER_ENLISTMENT_MTAG_COMMITREQ](#) message from its transaction manager to determine the outcome for the transaction.

4.5.1.2 Phase One - Subordinate Transaction Manager Facets

If the subordinate branch is a transaction manager (that is, it is using either a [CONNTYPE_PARTNERTM_BRANCH](#) or a [CONNTYPE_PARTNERTM_PROPAGATE](#) connection), the transaction manager sends a [PARTNERTM_PROPAGATE_MTAG_PREPAREREQ](#) user message that has **fSinglePhase** set to zero. If the connection was created by using [CONNTYPE_PARTNERTM_BRANCH](#), **fIsMaster** is zero (0). If the connection was created by using [CONNTYPE_PARTNERTM_PROPAGATE](#), **fIsMaster** is one (1).

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002003	PARTNERTM_PROPAGATE_MTAG_PREPAREREQ
dwcbVarLenData	0x00000008	8
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
grfRM	0x00000000	0
fSinglePhase	0x00000000	0

When the [subordinate transaction manager facet](#) receives the prepare request for a transaction, it then iterates through each of the transaction's subordinate branches and notifies the subordinates that the transaction processing has begun. The transaction manager waits for reply notifications from each of the subordinates in order to determine the outcome of the transaction.

If each subordinate branch of a transaction successfully prepares for the transaction (that is, each subordinate replies with a TXUSER_ENLISTMENT_PREPAREREQDONE_OK or PARTNERTM_PROPAGATE_PREPAREREQDONE_OK message depending on the connection type), the transaction manager replies to its [superior transaction manager facet](#) with a PARTNERTM_PROPAGATE_PREPAREREQDONE message that has **prepareReqDone** set to PARTNERTM_PROPAGATE_PREPAREREQDONE_OK. If the connection was created by using CONNTYPE_PARTNERTM_BRANCH, **fIsMaster** is one (1). If the connection was created by using CONNTYPE_PARTNERTM_PROPAGATE, **fIsMaster** is zero (0).

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002006	PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE
dwcbVarLenData	0x00000014	20
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
prepareReqDone	0x00000000	PARTNERTM_PROPAGATE_PREPAREREQDONE_OK
guidReason	0x00000000 0x00000000 0x00000000 0x00000000	00000000-0000-0000-0000-000000000000

The transaction manager now waits for a [PARTNERTM_PROPAGATE_MTAG_ABORTREQ](#) or [PARTNERTM_PROPAGATE_MTAG_COMMITREQ](#) message from its superior transaction manager facet to determine the outcome for the transaction.

4.5.1.3 Phase One - The Root Transaction Manager

If each subordinate branch of the root transaction manager successfully prepares for the transaction, that is, each subordinate replies with a TXUSER_ENLISTMENT_PREPAREREQDONE_OK or PARTNERTM_PROPAGATE_PREPAREREQDONE_OK message depending on the connection type, the root transaction manager replies to the application that the transaction has committed. It replies by sending a [TXUSER_BEGIN2_MTAG_SINK_ERROR](#) message with an Error value of TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED. For more information, see section [4.1.2](#).

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00006005	TXUSER_BEGIN2_MTAG_SINK_ERROR
dwcbVarLenData	0x00000004	4
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

Field	Value	Value description
Error	0x0000001F	TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED

The root transaction manager then initiates [Phase Two](#) processing.

4.5.2 Phase Two

The root transaction manager begins [Phase Two](#) by iterating through each subordinate branch of the transaction and notifying the subordinates that Phase Two processing has begun. In this example, the transaction commits.

4.5.2.1 Phase Two - Subordinate Resource Managers

If the subordinate branch is a resource manager (that is, it uses a [CONNTYPE_TXUSER_ENLISTMENT](#) connection), the transaction manager sends a [TXUSER_ENLISTMENT_MTAG_COMMITREQ](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00001035	TXUSER_ENLISTMENT_MTAG_COMMITREQ
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

When the resource manager successfully completes its commit work, it replies to its transaction manager with a [TXUSER_ENLISTMENT_MTAG_COMMITREQDONE](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00001038	TXUSER_ENLISTMENT_MTAG_COMMITREQDONE
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The resource manager has now completed all its work for the transaction and initiates the disconnect sequence on its CONNTYPE_TXUSER_ENLISTMENT connection with its transaction manager.

4.5.2.2 Phase Two - Subordinate Transaction Manager Facets

If the subordinate branch is a transaction manager (that is, it is using either a [CONNTYPE_PARTNERTM_BRANCH](#) or a [CONNTYPE_PARTNERTM_PROPAGATE](#) connection), the transaction manager sends a [PARTNERTM_PROPAGATE_MTAG_COMMITREQ](#) user message. If the connection was created by using [CONNTYPE_PARTNERTM_BRANCH](#), **fIsMaster** is zero (0). If the connection was created by using [CONNTYPE_PARTNERTM_PROPAGATE](#), **fIsMaster** is one (1).

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002005	PARTNERTM_PROPAGATE_MTAG_COMMITREQ
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

When the [Subordinate Transaction Manager Facet](#) receives the commit request for a transaction, it then iterates through each subordinate branch of the transaction and notifies the subordinates that the transaction is committed. The transaction manager then waits for reply notifications from each of the subordinates in order to complete [Phase Two](#) processing.

When each subordinate branch of the transaction replies that they have committed the transaction (that is, that each subordinate replies with a TXUSER_ENLISTMENT_COMMITREQDONE_OK or PARTNERTM_PROPAGATE_COMMITREQDONE_OK message, depending on the connection type), the transaction manager replies to its [Superior Transaction Manager Facet](#) with a PARTNERTM_PROPAGATE_COMMITREQDONE message. If the connection was created by using [CONNTYPE_PARTNERTM_BRANCH](#), **fIsMaster** is one (1). If the connection was created by using [CONNTYPE_PARTNERTM_PROPAGATE](#), then **fIsMaster** is zero (0).

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00002008	PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The subordinate transaction manager facet has now completed all the work that is associated with the transaction. If the subordinate transaction manager facet's connection with its superior transaction manager facet is a [CONNTYPE_PARTNERTM_BRANCH](#) connection, the subordinate transaction manager facet initiates the disconnect sequence. If the subordinate transaction manager facet has any [CONNTYPE_PARTNERTM_PROPAGATE](#) connections with its subordinate branches, the subordinate transaction manager facet initiates the disconnect sequence on those subordinate branch connections.

4.5.2.3 Phase Two - The Root Transaction Manager

After the root transaction manager receives all reply notifications from each of its subordinates, the transaction life cycle is complete. If the root transaction manager has any [CONNTYPE_PARTNERTM_PROPAGATE](#) connections with its subordinate branches, the root transaction manager initiates the disconnect sequence on those subordinate branch connections.

4.6 Resource Manager Recovery Scenario

This scenario describes in more detail how a resource manager registers with a transaction manager, and how the resource manager drives its recovery process. The scenario begins by the resource manager establishing a transport session with a transaction manager and negotiating its connection resources.

4.6.1 Initializing the Recovery Process

After the resource manager registers with the transaction manager (compare [Registering with the Transaction Manager as a Resource Manager](#)), it initiates recovery. To perform recovery, the resource manager iterates through its log and locates all in-doubt transactions and requests their outcome by reenlisting in the transaction with the transaction manager.

4.6.2 Reenlisting in In-Doubt Transactions

To reenlist in any transaction that is in-doubt, the resource manager establishes a [CONNTYPE_TXUSER_REENLIST](#) connection with its transaction manager.

CONNTYPE_TXUSER_REENLIST: The packet sequence starts when the resource manager initiates a CONNTYPE_TXUSER_REENLIST connection.

Field	Value	Value description
MsgTag	0x00000005	MTAG_CONNECTION_REQ
fIsMaster	0x00000001	1
dwConnectionId	0x00000002	2
dwUserMsgType	0x00000006	CONNTYPE_TXUSER_REENLIST
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

For each in-doubt transaction, the resource manager sends a [TXUSER_REENLIST MTAG_REENLIST](#) user message specifying the transaction identifier (guidTx), the time-out (in milliseconds) that it will wait for notification, and the resource manager identifier (guidRm). For this sample, the resource manager will wait one (1) second (or 1000 milliseconds).

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000002	2

Field	Value	Value description
dwUserMsgType	0x00001061	TXUSER_REENLIST_MTAG_REENLIST
dwcbVarLenData	0x00000024	36
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64
guidTx	0x4046037E 0x46C99722 0x06999883 0x35CB4123	4046037e-9722-46c9-9883-99062341cb35
ulTimeout	0x000003E8	1000
guidRm	0xE7BAEBDF 0x4E2BDC69 0xA1699FF1 0x772859D3	E7BAEBDF-DC69-4E2B-9FF1-69A1D3592877

When transaction manager receives the reenlist request, it attempts to find the transaction in its list of known transactions. If the transaction manager cannot locate the transaction, it assumes that the transaction aborted and replies to the resource manager with a [TXUSER_REENLIST_MTAG_REENLIST_ABORTED](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001062	TXUSER_REENLIST_MTAG_REENLIST_ABORTED
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

If the transaction manager can locate the transaction, the transaction manager attempts to determine outcome. The transaction manager replies to the resource manager with a [TXUSER_REENLIST_MTAG_REENLIST_COMMITTED](#) or [TXUSER_REENLIST_MTAG_REENLIST_ABORTED](#) user message, as appropriate.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001063	TXUSER_REENLIST_MTAG_REENLIST_COMMITTED
dwcbVarLenData	0x00000000	0

Field	Value	Value description
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

If the transaction manager is unable to determine outcome in the time-out period, the transaction manager replies to the resource manager with a [TXUSER_REENLIST_MTAG_REENLIST_TIMEOUT](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000002	2
dwUserMsgType	0x00001064	TXUSER_REENLIST_MTAG_REENLIST_TIMEOUT
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

After the resource manager determines the outcome from the transaction manager, it performs any remaining commit or abort work, as appropriate. If the resource manager receives a time-out notification, it needs to maintain the in-doubt entries in its log and is unable to determine outcome until the next time the resource manager performs recovery.

For any remaining in-doubt transactions, the resource manager needs to perform the previous steps for each in-doubt transaction.

If there are no more in-doubt transactions, the resource manager informs the transaction manager that it has completed its recovery process. The resource manager then initiates the disconnect sequence on this connection.

4.6.3 Completing Recovery

To complete recovery, the resource manager needs to send the transaction manager a [TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE](#) user message over its [CONNTYPE_TXUSER_RESOURCEMANAGER](#) connection.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000001	1
dwConnectionId	0x00000001	1
dwUserMsgType	0x00001052	TXUSER_RESOURCEMANAGER_MTAG_REENLISTMENTCOMPLETE
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

When the transaction manager receives the reenlistment complete notification, the transaction manager can clean up any transactions that are associated with the resource manager, such as the

transactions in the [Failed to Notify](#) state. In response, the transaction manager sends the resource manager a [TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE](#) user message.

Field	Value	Value description
MsgTag	0x00000FFF	MTAG_USER_MESSAGE
fIsMaster	0x00000000	0
dwConnectionId	0x00000001	1
dwUserMsgType	0x00001053	TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE
dwcbVarLenData	0x00000000	0
dwReserved1	0xcd64cd64	dwReserved1: 0xcd64cd64

The resource manager will maintain this connection.

5 Security

The following sections specify security considerations for implementers of the MSDTC Connection Manager: OleTx Transaction Protocol.

5.1 Security Considerations for Implementers

The transaction processing protocol that is defined by this specification is intended for use in an environment where all participants are trusted to collaborate in driving transactions toward a final outcome.

Misuse of the Two-Phase Commit Protocol can enable participants to perform simple denial of service attacks on their transaction managers. Because transaction managers generally communicate with multiple participants simultaneously, this condition represents a denial of service to other participants.

Consequently, implementers SHOULD take the following steps to ensure that transaction processing occurs in a secure environment:

- Each participant SHOULD initialize [\[MS-CMPO\]](#) sessions by using mutual authentication, as specified in [\[MS-CMPO\]](#) section 3.2.1.1.
- All transaction manager and resource manager implementations SHOULD uphold the following principles:
 - Every transaction reaches a common outcome for all participants, in accord with a correctly executed Two-Phase Commit Protocol.
- No transaction remains [In Doubt](#) for a longer period of time than the application's higher-layer business logic accepts. This particular determination is implementation-specific. [<52>](#)
- When authentication credentials are available, the acceptor SHOULD authorize incoming connections to ensure that the initiator is entitled to perform the actions that it is requesting. Implementations SHOULD adhere to the following authorization policies:

1. The following connection types SHOULD only be accepted for authenticated principals that have elevated privileges:

[CONNTYPE_TXUSER_RESOLVE](#)
[CONNTYPE_TXUSER_TRACE](#)

2. The following connection types SHOULD only be accepted for authenticated principals that are known to be transaction managers:

[CONNTYPE_PARTNERTM_PROPAGATE](#)
[CONNTYPE_PARTNERTM_REDELIVERCOMMIT](#)
[CONNTYPE_PARTNERTM_CHECKABORT](#)
[CONNTYPE_PARTNERTM_BRANCH](#)

3. Transaction manager implementations SHOULD ensure that the remote participant is a transaction manager for connection types that are used only between a superior transaction manager and a subordinate transaction manager.

An implementation can further restrict the set of supported connection types through configuration. These restrictions are reflected in the values of the **grfNetworkDtcAccess** and **grfXaTransactions** fields of the [TXUSER_GETSECURITYFLAGS_MTAG_FETCHED](#) message.

[<53>](#)

5.2 Index of Security Parameters

Security parameter	Section
RPC Security Level	[MS-CMPO] section 3.2.1.1
Transaction Manager Security Flags	3.2

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows NT 4.0 Option Pack
- Windows 2000
- Windows XP
- Windows Vista
- Windows Server 2003
- Windows Server 2008

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

[<1> Section 2.1.2.1:](#) No security is supported by the Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, Windows 2000, and Windows NT 4.0 Option Pack. Incoming authentication and mutual authentication are supported by Windows Server 2008; Windows Vista; Windows Server 2003 SP1 or later; and Windows XP SP2 and later. The security level is configurable to any of the three values on Windows Server 2008; Windows Vista; Windows Server 2003 SP1 or later; and Windows XP SP2 and later.

[<2> Section 2.1.2.3:](#) While performing push propagation, the Name object (as specified in [\[MS-CMPO\]](#) section 3.2.1.4) of the transaction manager is represented by using the [NAMEOBJECTBLOB](#) (section 2.2.5.3) structure on Windows NT 4.0 Option Pack and is represented by using the [SWhereabouts](#) (section 2.2.5.11) structure on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

[<3> Section 2.2.1.1.1:](#) The connection type CONNTYPE_TXUSER_EXTENDEDWHEREABOUTS is supported by Windows Vista; Windows XP SP2 and later or later; and Windows XP SP3 and later.

[<4> Section 2.2.1.1.1:](#) The connection type CONNTYPE_TXUSER_PROMOTE is supported by Windows Server 2008 and Windows Vista.

[<5> Section 2.2.1.1.1:](#) The connection type CONNTYPE_TXUSER_SETTXTIMEOUT is supported by Windows 2000 SP4 or later.

[<6> Section 2.2.1.1.1:](#) The connection type CONNTYPE_TXUSER_SETTXTIMEOUT2 is supported by Windows Server 2008, Windows Vista, and Windows Server 2003.

[<7> Section 2.2.1.1.1:](#) The message TXUSER_RESOLVE_MTAG_ACCESSDENIED that is associated with connection type CONNTYPE_TXUSER_RESOLVE is supported by Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP SP2.

[<8> Section 2.2.1.1.3:](#) Connection type CONNTYPE_TXUSER_RESOURCEMANAGERINTERNAL is supported by Windows Vista and Windows Server 2008.

[<9> Section 2.2.4.1:](#) All versions of Windows set this field to a non-deterministic 4-byte value.

[<10> Section 2.2.5.2:](#) This structure is supported only on Windows XP.

[<11> Section 2.2.5.3:](#) On Windows NT 4.0 Option Pack and Windows 2000, this field is set to a non-deterministic 4-byte value. Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 set this field to 0xCD64CD64.

[<12> Section 2.2.5.4:](#) The **dwVersionMax** field is set to 1 on Windows NT 4.0 Option Pack; 2 on Windows 2000 and 3 on Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008.

[<13> Section 2.2.5.4:](#) All versions of Windows supported by this protocol accept a string from the application for the value of the field **szDesc** and pass it along unmodified and uninterpreted.

[<14> Section 2.2.5.9:](#) The field **TmProtocolMsdtcV1** is included in the [SWhereabouts](#) structure on Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, Windows 2000, and Windows NT 4.0 Option Pack. The field TmProtocolMsdtcV2 is included in the [SWhereabouts](#) structure on Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, and Windows 2000. The field TmProtocolTip is included in the [SWhereabouts](#) structure, if the transaction manager is so configured, on Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, and Windows 2000. The field TmProtocolExtended is included in the [SWhereabouts](#) structure, if the transaction manager is so configured, on Windows Server 2008; Windows Vista; Windows Server 2003 SP2 or later; and Windows XP SP3 or later.

[<15> Section 2.2.5.11:](#) Windows initializes these padding bytes with random values.

[<16> Section 2.2.6.8:](#) The values of the OLETX_ISOLATION_LEVEL enumeration are not interpreted by the transaction manager. They are typically interpreted by resource managers that implement data isolation. These values are transported by the transaction manager from the root application to the resource managers.

[<17> Section 2.2.7.1:](#) All versions of Windows supported by this protocol accept the value of grfRM from the higher layer software with no interpretation and pass it along unmodified and uninterpreted.

[<18> Section 2.2.7.2:](#) RPC connections established by the sending MSDTC instance are first attempted over an authenticated connection, and if that fails, over an unauthenticated connection.

[<19> Section 2.2.8.1.1.1:](#) All versions of Windows that are supported by this protocol set the value of the **guidReason** field to a NULL GUID and ignore it on receipt.

[<20> Section 2.2.8.1.1.2:](#) All versions of Windows that are supported by this protocol accept the value for the field **isoLevel** from the higher-layer software and pass it along unmodified and uninterpreted.

[<21> Section 2.2.8.1.1.2:](#) All versions of Windows supported by this protocol accept the value for the **szDesc** field from the higher-layer software and pass it along unmodified and uninterpreted.

[<22> Section 2.2.8.2.1.1.10:](#) All versions of Windows supported by this protocol have a limit of 32 local participants.

[<23> Section 2.2.8.2.1.1.11:](#) All versions of Windows that are supported by this protocol have a limit of 32 remote participants.

[<24> Section 2.2.8.2.2.2.10:](#) All versions of Windows supported by this protocol have a limit of 32 remote participants.

[<25> Section 2.2.8.2.2.3.1:](#) All versions of Windows supported by this protocol accept the value for the field **guidReason** from the higher-layer software and pass it along unmodified and uninterpreted.

<26> [Section 2.2.8.3.1.2](#): Windows initializes these padding bytes with random values.

<27> [Section 2.2.8.3.1.2](#): Windows initializes these padding bytes with random values.

<28> [Section 2.2.8.3.1.2](#): For OleTx participants, the name is the resource manager GUID in string form (as specified in [\[C706\]](#), [Appendix A, UUID](#)), and the identifier is zero.

<29> [Section 2.2.8.3.1.2](#): Windows initializes these padding bytes with random values.

<30> [Section 2.2.8.3.2.1](#): By default on Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP SP2 and later, Windows requires authentication with an account that is in the administrator group. This behavior is configurable on Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP SP2 and later.

<31> [Section 2.2.8.3.5](#): All versions of Windows that implement this protocol store the data in a local trace file. For more information, see [\[MSDN-DTCTrace\]](#).

<32> [Section 2.2.10.2.2.8](#): All versions of Windows supported by this protocol limit transactions to 32 direct enlistments.

<33> [Section 2.2.10.2.2.12](#): All versions of Windows that are supported this protocol receive the value for the **guidReason** field from higher-layer software. It makes no attempt to interpret the value but does attempt to supply this value back to the application software on a transaction abort.

<34> [Section 3.1.4.3](#): Regarding the MSDTC Connection Manager: OleTx Transaction Protocol connection establishment in Windows, an MSDTC Connection Manager: OleTx Transaction Protocol session partner does send connection requests for connection types that it supports (when required by the protocol rules) (see section 3), but that are not supported by the negotiated protocol version except for two connection types: [CONNTYPE_TXUSER_IMPORT2 \(section 2.2.8.2.2.4\)](#) and [CONNTYPE_TXUSER_PROMOTE \(section 2.2.8.1.3\)](#). As a result, the requests for the unsupported connection types are rejected with a [MTAG_CONNECTION_REQ_DENIED](#) (see [\[MS-CMP\]](#) section 2.2.5 MsgTag 0x00000005), and the partner falls back to an alternative supported connection type that is supported by the negotiated protocol version.

<35> [Section 3.1.4.3](#): Regarding the MSDTC Connection Manager: OleTx Transaction Protocol connection establishment in Windows, an MSDTC Connection Manager: OleTx Transaction Protocol session partner does accept as valid connection requests for connection types that it supports, but that are not supported by the negotiated protocol version. This situation can only occur if its interlocutor partner is not an MSDTC Connection Manager: OleTx Transaction Protocol implementation.

<36> [Section 3.1.4.3](#): Regarding the sending of messages over an established MSDTC Connection Manager: OleTx Transaction Protocol connection in Windows, an MSDTC Connection Manager: OleTx Transaction Protocol session partner never sends messages that it supports (when required by the protocol rules) (see section 3), but that are not supported by the negotiated protocol version (in the context of the connection's connection type) with one exception: [TXUSER_RESOLVE_MTAG_ACCESSDENIED \(section 2.2.8.3.2.1\)](#). For [TXUSER_RESOLVE_MTAG_ACCESSDENIED](#), a partner that supports this message sends it (when required by protocol rules) even if it is not supported by the negotiated protocol version.

<37> [Section 3.1.4.3](#): Regarding receiving messages over an established MSDTC Connection Manager: OleTx Transaction Protocol connection in Windows, an MSDTC Connection Manager: OleTx Transaction Protocol session partner does accept as valid messages that it supports (in the context of the connection's connection type), but that are not supported by the negotiated protocol version. This situation can only occur if the interlocutor partner is not an MSDTC Connection Manager: OleTx Transaction Protocol implementation.

[<38> Section 3.1.6:](#) On receiving an invalid message on a connection, the participant terminates the associated session on Windows XP SP1, Windows 2000, and Windows NT 4.0 Option Pack.

[<39> Section 3.2.1:](#) The incoming connection is refused if the identity is not established. Also, for connection types CONNTYPE_PARTNERTM_PROPAGATE, CONNTYPE_PARTNERTM_REDELIVERCOMMIT, CONNTYPE_PARTNERTM_CHECKABORT, and CONNTYPE_PARTNERTM_BRANCH, the incoming identity MUST match the pattern "<domain>\<incoming-MSDTC-name>\$", where <incoming-MSDTC-name> is the source hostname for the connection, and <domain> is the name of the domain in which the host is a member. RPC connections that are established by the sending MSDTC instance are attempted only over an authenticated connection.

[<40> Section 3.2.1.5:](#) All versions of Windows that implement this protocol use in-process calls to exchange events between facets.

[<41> Section 3.2.2.1:](#) On Windows platforms, if no timeout value is provided when creating a transaction (see [Beginning a Transaction \(section 3.3.4.1\)](#)), the transaction's timeout timer is set to zero.

[<42> Section 3.2.3.1:](#) On Windows XP SP1, Windows 2000, and Windows NT 4.0 Option Pack, the security level is initialized to No Security and MUST not be configured to any other value. On Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP SP2 and later, the security level is initialized to mutual authentication and is configurable to any of the three values.

[<43> Section 3.2.3.1:](#) On Windows XP SP1, Windows 2000, and Windows NT 4.0 Option Pack, each of the following security flags (with the exception of Allow TIP and Allow XA) is initialized to true and MUST not be configured to any other value. The flags Allow TIP and Allow XA are not used on Windows NT 4.0 Option Pack; these flags are initialized to true and MUST not be configured to any other value on Windows XP SP1 and Windows 2000. On Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP SP2 and later, each of the following security flags is initialized to false and MUST be configured to either of the two values.

[<44> Section 3.2.3.2:](#) The Timeout value defaults to the value zero on Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, Windows 2000, and Windows NT 4.0 Option Pack.

[<45> Section 3.2.7.11:](#) The limit of Subordinate Enlistments depends on the type of the enlistment. The limit on Subordinate Transaction Manager Enlistments is 16 on Windows NT 4.0 Option Pack; and 64 on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. The limit on Subordinate Resource Manager Enlistments is 32 on Windows NT 4.0 Option Pack, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

[<46> Section 3.2.7.13:](#) The [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) ensures that transactions with duplicate identifiers will not be created on Windows NT 4.0 Option Pack, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2008; however, the [Core Transaction Manager Facet \(section 1.3.3.3.1\)](#) does not ensure the same on Windows Vista.

[<47> Section 3.4.5.1.3.1:](#) The Windows Vista implementation accepts this message in the Active state, and processes it as if it were in the Idle state.

[<48> Section 3.4.7.7:](#) For Windows Vista, if a duplicate Transaction Identifier is found, the Error field of [TXUSER_BEGIN2_MTAG_SINK_ERROR](#) is set to the value TRUN_TXBEGIN_ERROR_BEGIN_LOG_FULL.

[<49> Section 3.5.4.10:](#) The session identifier is set to the session identifier of the underlying transport on Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, Windows 2000, and Windows NT 4.0 Option Pack.

[<50> Section 3.7.2.1:](#) The value of the [Redeliver Commit Timer](#) is 1,000 milliseconds if the underlying transport is down. Otherwise, it is 500 milliseconds on Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, Windows 2000, and Windows NT 4.0 Option Pack. This value is not configurable.

[<51> Section 3.8.2.1:](#) The value of the Check Abort timer is 1,000 milliseconds on Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, Windows 2000, and Windows NT 4.0 Option Pack. This value is not configurable.

[<52> Section 5.1:](#) Mutual authentication is used by default on Windows XP SP2 and later, Windows Server 2003, Windows Vista, and Windows Server 2008. No authentication is used on Windows NT 4.0 Option Pack, Windows 2000, and Windows XP SP1.

[<53> Section 5.1:](#) All Windows implementations perform the following authorization steps on incoming connections:

- When mutual authentication is required, the following connection types MUST be established by a user identity whose principal name takes the form of <DomainName>\<MachineName>\$ where <DomainName> is a NetBIOS domain name and <MachineName> matches the NetBIOS host name of the machine initiating the connection:
 - [CONNTYPE PARTNERTM PROPAGATE](#)
 - [CONNTYPE PARTNERTM REDELIVERCOMMIT](#)
 - [CONNTYPE PARTNERTM CHECKABORT](#)
 - [CONNTYPE PARTNERTM BRANCH](#)
- When incoming authentication is available, the following connection types MUST be established by a user identity that is authenticated as an administrator:
 - [CONNTYPE TXUSER RESOLVE](#)
 - [CONNTYPE TXUSER TRACE](#)

7 Index

A

Abstract data model

application ([section 3.1.1](#), [section 3.3.1](#), [section 3.4.1](#))

core transaction manager ([section 3.1.1](#), [section 3.2.1](#))

resource manager ([section 3.1.1](#), [section 3.5.1](#), [section 3.6.1](#))

subordinate transaction manager ([section 3.1.1](#), [section 3.8.1](#))

superior transaction manager ([section 3.1.1](#), [section 3.7.1](#))

transaction manager ([section 3.1.1](#), [section 3.4.1](#), [section 3.6.1](#))

[Applicability](#)

Application

abstract data model ([section 3.1.1](#), [section 3.3.1](#), [section 3.4.1](#))

[connection types](#)

[facet - role](#)

higher-layer triggered events ([section 3.1.5](#), [section 3.3.4](#), [section 3.4.4](#))

initialization ([section 3.1.3](#), [section 3.3.3](#), [section 3.4.3](#))

local events ([section 3.1.8](#), [section 3.3.7](#), [section 3.4.7](#))

message processing ([section 3.1.6](#), [section 3.3.5](#), [section 3.4.5](#))

overview ([section 3.1](#), [section 3.3](#), [section 3.4](#))

[role](#)

sequencing rules ([section 3.1.6](#), [section 3.3.5](#), [section 3.4.5](#))

timer events ([section 3.1.7](#), [section 3.3.6](#), [section 3.4.6](#))

timers ([section 3.1.2](#), [section 3.3.2](#), [section 3.4.2](#))

[versioning](#)

[Associate Msg Version2 packet](#)

[Associate Msg Version3 packet](#)

C

[Capability negotiation](#)

[Capability negotiation mechanisms](#)

Connection types

[application](#)

[resource manager](#)

[resource manager - transaction recovery](#)

[resource manager - voting](#)

[transaction administration](#)

[transaction manager](#)

[transaction manager propagation](#)

[transaction propagation](#)

[transaction recovery](#)

[Connections](#)

[CONNTYPE enumeration](#)

[CONNTYPE PARTNERTM BRANCH](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE PARTNERTM CHECKABORT](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE PARTNERTM PROPAGATE](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE PARTNERTM REDELIVERCOMMIT](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER ASSOCIATE](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER BEGIN2](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER BEGINNER](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER ENLISTMENT](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER EXPORT](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER EXTENDEDWHEREABOUTS](#)

[acceptor states](#)

[CONNTYPE TXUSER_EXTENDWHEREABOUTS](#)

[initiator states](#)

[CONNTYPE TXUSER GETSECURITYFLAGS](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER GETTXDETAILS](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER IMPORT](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER IMPORT2](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER PHASE0](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER PROMOTE](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER REENLIST](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER RESOLVE](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER RESOURCEMANAGER](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER RESOURCEMANAGERINTERNAL](#)

[acceptor states](#)

[initiator states](#)

[CONNTYPE TXUSER SETTXTIMEOUT](#)

[acceptor states](#)
[initiator states](#)
[CONNTYPE_TXUSER_SETTXTIMEOUT2](#)
[acceptor states](#)
[initiator states](#)
[CONNTYPE_TXUSER_TRACE](#)
[acceptor states](#)
[initiator states](#)
[CONNTYPE_TXUSER_VOTER](#)
[acceptor states](#)
[initiator states](#)
[Constants](#)

Core transaction manager
 abstract data model ([section 3.1.1](#), [section 3.2.1](#))
 higher-layer triggered events ([section 3.1.5](#), [section 3.2.4](#))
 initialization ([section 3.1.3](#), [section 3.2.3](#))
 local events ([section 3.1.8](#), [section 3.2.7](#))
 message processing ([section 3.1.6](#), [section 3.2.5](#))
 overview ([section 3.1](#), [section 3.2](#))
 role
 sequencing rules ([section 3.1.6](#), [section 3.2.5](#))
 timer events ([section 3.1.7](#), [section 3.2.6](#))
 timers ([section 3.1.2](#), [section 3.2.2](#))
 versioning

D

Data model - abstract
 application ([section 3.1.1](#), [section 3.3.1](#), [section 3.4.1](#))
 core transaction manager ([section 3.1.1](#), [section 3.2.1](#))
 resource manager ([section 3.1.1](#), [section 3.5.1](#), [section 3.6.1](#))
 subordinate transaction manager ([section 3.1.1](#), [section 3.8.1](#))
 superior transaction manager ([section 3.1.1](#), [section 3.7.1](#))
 transaction manager ([section 3.1.1](#), [section 3.4.1](#), [section 3.6.1](#))
[DTCADVCONFIG packet](#)
[Durability](#)

E

[Enlistment example](#)
[Enlistment objects - core transaction manager](#)
[Enumerations](#)
Examples
 [overview](#)
 [resource manager recovery scenario](#)
 [simple enlistment scenario](#)
 [simple transaction scenario](#)
 [transaction manager two-phase commit scenario](#)
 transaction marshaling scenario ([section 4.2](#), [section 4.3](#))

F

[Fields - vendor-extensible](#)

G

[Glossary](#)
[GRFRM](#)

H

Higher-layer triggered events
 Application ([section 3.1.5](#), [section 3.3.4](#), [section 3.4.4](#))
 Core transaction manager ([section 3.1.5](#), [section 3.2.4](#))
 Resource manager ([section 3.1.5](#), [section 3.5.4](#), [section 3.6.4](#))
 Subordinate transaction manager ([section 3.1.5](#), [section 3.8.4](#))
 Superior transaction manager ([section 3.1.5](#), [section 3.7.4](#))
 Transaction manager ([section 3.1.5](#), [section 3.4.4](#), [section 3.6.4](#))
 [Transaction manager - core](#)
 [Transaction manager - subordinate](#)
 [Transaction manager - superior](#)

I

[Implementers - security considerations](#)
[Informative references](#)

Initialization
 application ([section 3.1.3](#), [section 3.3.3](#), [section 3.4.3](#))
 core transaction manager ([section 3.1.3](#), [section 3.2.3](#))
 resource manager ([section 3.1.3](#), [section 3.5.3](#), [section 3.6.3](#))
 subordinate transaction manager ([section 3.1.3](#), [section 3.8.3](#))
 superior transaction manager ([section 3.1.3](#), [section 3.7.3](#))
 transaction manager ([section 3.1.3](#), [section 3.4.3](#), [section 3.6.3](#))

[Introduction](#)

L

[Lifetime - transaction](#)
Local events
 application ([section 3.1.8](#), [section 3.3.7](#), [section 3.4.7](#))
 core transaction manager ([section 3.1.8](#), [section 3.2.7](#))
 resource manager ([section 3.1.8](#), [section 3.5.7](#), [section 3.6.7](#))
 subordinate transaction manager ([section 3.1.8](#), [section 3.8.7](#))
 superior transaction manager ([section 3.1.8](#), [section 3.7.7](#))
 transaction manager ([section 3.1.8](#), [section 3.4.7](#), [section 3.6.7](#))
[Logging - core transaction manager](#)

M

Message processing

- application ([section 3.1.6](#), [section 3.3.5](#), [section 3.4.5](#))
- core transaction manager ([section 3.1.6](#), [section 3.2.5](#))
- resource manager ([section 3.1.6](#), [section 3.5.5](#), [section 3.6.5](#))
- subordinate transaction manager ([section 3.1.6](#), [section 3.8.5](#))
- superior transaction manager ([section 3.1.6](#), [section 3.7.5](#))
- transaction manager ([section 3.1.6](#), [section 3.4.5](#), [section 3.6.5](#))

[MESSAGE PACKET packet](#)

Messages

- [overview](#)
- [syntax](#)
- [transport](#)

[MS-CMPO parameterization](#)

N

[Name Object - computing](#)

[NAMEOBJECTBLOB packet](#)

[Normative references](#)

O

[OLETX ISOLATION FLAGS enumeration](#)

[OLETX ISOLATION LEVEL enumeration](#)

[OLETX_TM_ADDR packet](#)

[OLETX_VARLEN_STRING packet](#)

[Overview](#)

P

[Parameters - security](#)

[PARTNERTM_BRANCH_MTAG_BRANCH_LOG_FULL packet](#)

[PARTNERTM_BRANCH_MTAG_BRANCH_NO_MEM packet](#)

[PARTNERTM_BRANCH_MTAG_BRANCH_TOO_LATE packet](#)

[PARTNERTM_BRANCH_MTAG_BRANCH_TOO_MANY packet](#)

[PARTNERTM_BRANCH_MTAG_BRANCH_TX_NOT_FOUND packet](#)

[PARTNERTM_BRANCH_MTAG_BRANCHED packet](#)

[PARTNERTM_BRANCH_MTAG_BRANCHING packet](#)

[PARTNERTM_CHECKABORT_MTAG_ABORTED packet](#)

[PARTNERTM_CHECKABORT_MTAG_CHECK packet](#)

[PARTNERTM_CHECKABORT_MTAG_RETRY packet](#)

[PARTNERTM_PROPAGATE_MTAG_ABORTNOTIFY packet](#)

[PARTNERTM_PROPAGATE_MTAG_ABORTREQ packet](#)

[PARTNERTM_PROPAGATE_MTAG_ABORTREQDONE packet](#)

[PARTNERTM_PROPAGATE_MTAG_COMMITREQ packet](#)

[PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE packet](#)

[PARTNERTM_PROPAGATE_MTAG_DUPLICATE packet](#)

[PARTNERTM_PROPAGATE_MTAG_LOG_FULL packet](#)

[PARTNERTM_PROPAGATE_MTAG_NO_MEM packet](#)

[PARTNERTM_PROPAGATE_MTAG_PHASE0 packet](#)

[PARTNERTM_PROPAGATE_MTAG_PHASE0COMPLETE packet](#)

[PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTER packet](#)

[PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTERED packet](#)

[PARTNERTM_PROPAGATE_MTAG_PHASE0REGISTRATIONREJECTED packet](#)

[PARTNERTM_PROPAGATE_MTAG_PREPAREREQ packet](#)

[PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE packet](#)

[PARTNERTM_PROPAGATE_MTAG_PROPAGATE packet](#)

[PARTNERTM_PROPAGATE_MTAG_PROPAGATED packet](#)

[PARTNERTM_PROPAGATE_MTAG_PROTOCOL_ERROR packet](#)

[PARTNERTM_PROPAGATE_PREPAREREQDONE_RESPONSE enumeration](#)

[PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQ packet](#)

[PARTNERTM_REDELIVERCOMMIT_MTAG_COMMITREQDONE packet](#)

[PARTNERTM_REDELIVERCOMMIT_MTAG_RETRY packet](#)

[Phase One](#)

[Phase Two](#)

[Phase Zero](#)

[Preconditions](#)

[Prerequisites](#)

Propagation

[pull \(section 1.3.5.1, section 2.2.8.2.1, section 2.2.9.1.2, section 4.2\)](#)

[push \(section 1.3.5.2, section 2.2.8.2.2, section 2.2.9.1.1, section 4.3\)](#)

[transaction \(section 1.3.5, section 2.2.8.2\)](#)

[transaction manager](#)

[Propagation Token packet](#)

[Pull propagation \(section 1.3.5.1, section 2.2.8.2.1, section 2.2.9.1.2, section 4.2\)](#)

[Push propagation \(section 1.3.5.2, section 2.2.8.2.2, section 2.2.9.1.1, section 4.3\)](#)

R

[Recovery example](#)

References

[informative](#)

[normative](#)

[overview](#)

[Registration - resource manager](#)

[Relationship to other protocols](#)

Resource manager

[abstract data model \(section 3.1.1, section 3.5.1, section 3.6.1\)](#)

[connection types](#)

[example](#)

[facet - role](#)

[higher-layer triggered events \(section 3.1.5, section 3.5.4, section 3.6.4\)](#)

[initialization \(section 3.1.3, section 3.5.3, section 3.6.3\)](#)
[local events \(section 3.1.8, section 3.5.7, section 3.6.7\)](#)
[message processing \(section 3.1.6, section 3.5.5, section 3.6.5\)](#)
[overview \(section 3.1, section 3.5, section 3.6\)](#)
[recovery](#)
[registration](#)
[role](#)
[sequencing rules \(section 3.1.6, section 3.5.5, section 3.6.5\)](#)
[timer events \(section 3.1.7, section 3.5.6, section 3.6.6\)](#)
[timers \(section 3.1.2, section 3.5.2, section 3.6.2\)](#)
[transaction coordination](#)
[versioning](#)
[voting](#)
[Resource manager recovery scenario](#)
[Roles - transaction](#)

S

[SDtcCmEndpointInfoV1 packet](#)
[SDtcCmEndpointInfoV2 packet](#)
[Security](#)
[computing levels](#)
 Sequencing rules
 [application \(section 3.1.6, section 3.3.5, section 3.4.5\)](#)
 [core transaction manager \(section 3.1.6, section 3.2.5\)](#)
 [resource manager \(section 3.1.6, section 3.5.5, section 3.6.5\)](#)
 [subordinate transaction manager \(section 3.1.6, section 3.8.5\)](#)
 [superior transaction manager \(section 3.1.6, section 3.7.5\)](#)
 [transaction manager \(section 3.1.6, section 3.4.5, section 3.6.5\)](#)
[Sessions](#)
[SExtendedEndpointInfo packet](#)
[Simple enlistment scenario](#)
[Simple transaction scenario](#)
[Single-phase commit](#)
[SOleTxInfoForTip packet](#)
[Standards assignments](#)
[STmToTmProtocol packet](#)
 Structures
 [common](#)
 [transaction propagation](#)
[STxInfo packet](#)
 Subordinate transaction manager
 [abstract data model \(section 3.1.1, section 3.8.1\)](#)
 [facet - role](#)
 [higher-layer triggered events \(section 3.1.5, section 3.8.4\)](#)
 [initialization \(section 3.1.3, section 3.8.3\)](#)
 [local events \(section 3.1.8, section 3.8.7\)](#)
 [message processing \(section 3.1.6, section 3.8.5\)](#)
 [overview \(section 3.1, section 3.8\)](#)

[sequencing rules \(section 3.1.6, section 3.8.5\)](#)
[timer events \(section 3.1.7, section 3.8.6\)](#)
[timers \(section 3.1.2, section 3.8.2\)](#)
[versioning](#)
[Subordinate-driven transaction recovery](#)
 Superior transaction manager
 [abstract data model \(section 3.1.1, section 3.7.1\)](#)
 [facet - role](#)
 [higher-layer triggered events \(section 3.1.5, section 3.7.4\)](#)
 [initialization \(section 3.1.3, section 3.7.3\)](#)
 [local events \(section 3.1.8, section 3.7.7\)](#)
 [message processing \(section 3.1.6, section 3.7.5\)](#)
 [overview \(section 3.1, section 3.7\)](#)
 [sequencing rules \(section 3.1.6, section 3.7.5\)](#)
 [timer events \(section 3.1.7, section 3.7.6\)](#)
 [timers \(section 3.1.2, section 3.7.2\)](#)
 [versioning](#)
[Superior-driven transaction recovery](#)
[SWhereabouts packet](#)
[Syntax - message](#)

T

Timer events
 [Application \(section 3.1.7, section 3.3.6, section 3.4.6\)](#)
 [Core transaction manager \(section 3.1.7, section 3.2.6\)](#)
 [Resource manager \(section 3.1.7, section 3.5.6, section 3.6.6\)](#)
 [Subordinate transaction manager \(section 3.1.7, section 3.8.6\)](#)
 [Superior transaction manager \(section 3.1.7, section 3.7.6\)](#)
 [Transaction manager \(section 3.1.7, section 3.4.6, section 3.6.6\)](#)
 [Transaction manager - core](#)
 [Transaction manager - subordinate](#)
 [Transaction manager - superior](#)
 Timers
 [Application \(section 3.1.2, section 3.3.2, section 3.4.2\)](#)
 [Core transaction manager \(section 3.1.2, section 3.2.2\)](#)
 [Resource manager \(section 3.1.2, section 3.5.2, section 3.6.2\)](#)
 [Subordinate transaction manager \(section 3.1.2, section 3.8.2\)](#)
 [Superior transaction manager \(section 3.1.2, section 3.7.2\)](#)
 [Transaction manager \(section 3.1.2, section 3.4.2, section 3.6.2\)](#)
 [Transaction manager - core](#)
 [Transaction manager - subordinate](#)
 [Transaction manager - superior](#)
[TM PROTOCOL enumeration](#)
 Transaction
 [administration - connection types](#)
 [completion](#)
 [constants](#)

- [enumerations](#)
- [initiation](#)
- [lifetime](#)
- [logging - core transaction manager](#)
- [manager administration](#)
- [marshaling - example \(section 4.2, section 4.3\)](#)
- [propagation](#)
- [propagation - connection types](#)
- [propagation - structures](#)
- [recovery](#)
- [roles](#)
- [simple - example](#)
- [states - core transaction manager](#)
- Transaction manager
 - abstract data model ([section 3.1.1](#), [section 3.4.1](#), [section 3.6.1](#))
 - [connection types](#)
 - higher-layer triggered events ([section 3.1.5](#), [section 3.4.4](#), [section 3.6.4](#))
 - initialization ([section 3.1.3](#), [section 3.4.3](#), [section 3.6.3](#))
 - local events ([section 3.1.8](#), [section 3.4.7](#), [section 3.6.7](#))
 - message processing ([section 3.1.6](#), [section 3.4.5](#), [section 3.6.5](#))
 - overview ([section 3.1](#), [section 3.4](#), [section 3.6](#))
 - [propagation - connection types](#)
 - [recovery](#)
 - [role](#)
 - sequencing rules ([section 3.1.6](#), [section 3.4.5](#), [section 3.6.5](#))
 - timer events ([section 3.1.7](#), [section 3.4.6](#), [section 3.6.6](#))
 - timers ([section 3.1.2](#), [section 3.4.2](#), [section 3.6.2](#))
 - [two-phase commit example](#)
 - [versioning](#)
- Transaction manager - core
 - abstract data model ([section 3.1.1](#), [section 3.2.1](#))
 - higher-layer triggered events ([section 3.1.5](#), [section 3.2.4](#))
 - initialization ([section 3.1.3](#), [section 3.2.3](#))
 - local events ([section 3.1.8](#), [section 3.2.7](#))
 - message processing ([section 3.1.6](#), [section 3.2.5](#))
 - overview ([section 3.1](#), [section 3.2](#))
 - sequencing rules ([section 3.1.6](#), [section 3.2.5](#))
 - timer events ([section 3.1.7](#), [section 3.2.6](#))
 - timers ([section 3.1.2](#), [section 3.2.2](#))
 - [versioning](#)
- Transaction manager - subordinate
 - abstract data model ([section 3.1.1](#), [section 3.8.1](#))
 - higher-layer triggered events ([section 3.1.5](#), [section 3.8.4](#))
 - initialization ([section 3.1.3](#), [section 3.8.3](#))
 - local events ([section 3.1.8](#), [section 3.8.7](#))
 - message processing ([section 3.1.6](#), [section 3.8.5](#))
 - overview ([section 3.1](#), [section 3.8](#))
 - sequencing rules ([section 3.1.6](#), [section 3.8.5](#))
 - timer events ([section 3.1.7](#), [section 3.8.6](#))
 - timers ([section 3.1.2](#), [section 3.8.2](#))
 - [versioning](#)
- Transaction manager - superior
 - abstract data model ([section 3.1.1](#), [section 3.7.1](#))
- higher-layer triggered events ([section 3.1.5](#), [section 3.7.4](#))
- initialization ([section 3.1.3](#), [section 3.7.3](#))
- local events ([section 3.1.8](#), [section 3.7.7](#))
- message processing ([section 3.1.6](#), [section 3.7.5](#))
- overview ([section 3.1](#), [section 3.7](#))
- sequencing rules ([section 3.1.6](#), [section 3.7.5](#))
- timer events ([section 3.1.7](#), [section 3.7.6](#))
- timers ([section 3.1.2](#), [section 3.7.2](#))
- [versioning](#)
- [Transaction manager administration](#)
- Transaction recovery
 - [connection types](#)
 - [resource manager - connection types](#)
- [Transport - message](#)
- Triggered events - higher-layer
 - Application ([section 3.1.5](#), [section 3.3.4](#), [section 3.4.4](#))
 - Core transaction manager ([section 3.1.5](#), [section 3.2.4](#))
 - Resource manager ([section 3.1.5](#), [section 3.5.4](#), [section 3.6.4](#))
 - Subordinate transaction manager ([section 3.1.5](#), [section 3.8.4](#))
 - Superior transaction manager ([section 3.1.5](#), [section 3.7.4](#))
 - Transaction manager ([section 3.1.5](#), [section 3.4.4](#), [section 3.6.4](#))
 - [Transaction manager - core](#)
 - [Transaction manager - subordinate](#)
 - [Transaction manager - superior](#)
 - [TRUN TXBEGIN ERRORS enumeration](#)
 - [TRUN TXIMPORT ERRORS enumeration](#)
 - [Two-phase commit example](#)
 - [TXUSER ASSOCIATE MTAG ASSOCIATE packet](#)
 - [TXUSER ASSOCIATE MTAG ASSOCIATED packet](#)
 - [TXUSER ASSOCIATE MTAG COMM FAILED packet](#)
 - [TXUSER ASSOCIATE MTAG CREATE BAD TMADDR packet](#)
 - [TXUSER ASSOCIATE MTAG LOG FULL LOCAL packet](#)
 - [TXUSER ASSOCIATE MTAG LOG FULL REMOTE packet](#)
 - [TXUSER ASSOCIATE MTAG NO MEM LOCAL packet](#)
 - [TXUSER ASSOCIATE MTAG NO MEM REMOTE packet](#)
 - [TXUSER ASSOCIATE MTAG TOO LATE packet](#)
 - [TXUSER ASSOCIATE MTAG TOO MANY LOCAL packet](#)
 - [TXUSER ASSOCIATE MTAG TOO MANY REMOTE packet](#)
 - [TXUSER ASSOCIATE MTAG TX NOT FOUND packet](#)
 - [TXUSER BEGIN2 MTAG ABORT packet](#)
 - [TXUSER BEGIN2 MTAG BEGIN packet](#)
 - [TXUSER BEGIN2 MTAG COMMIT packet](#)
 - [TXUSER BEGIN2 MTAG SINK BEGUN packet](#)
 - [TXUSER BEGIN2 MTAG SINK ERROR packet](#)
 - [TXUSER BEGINNER MTAG ABORT packet](#)
 - [TXUSER BEGINNER MTAG BEGIN packet](#)
 - [TXUSER BEGINNER MTAG BEGIN LOG FULL packet](#)
 - [TXUSER BEGINNER MTAG BEGIN NO MEM packet](#)
 - [TXUSER BEGINNER MTAG BEGUN packet](#)
 - [TXUSER BEGINNER MTAG COMMIT packet](#)
 - [TXUSER BEGINNER MTAG COMMIT INDOUBT packet](#)
 - [TXUSER BEGINNER MTAG COMMIT TOO LATE packet](#)

[TXUSER BEGINNER MTAG PROMOTE packet](#)
[TXUSER BEGINNER MTAG REQUEST COMPLETED packet](#)
[TXUSER ENLISTMENT MTAG ABORTREQ packet](#)
[TXUSER ENLISTMENT MTAG ABORTREQDONE packet](#)
[TXUSER ENLISTMENT MTAG COMMITREQ packet](#)
[TXUSER ENLISTMENT MTAG COMMITREQDONE packet](#)
[TXUSER ENLISTMENT MTAG ENLIST packet](#)
[TXUSER ENLISTMENT MTAG ENLIST LOG FULL packet](#)
[TXUSER ENLISTMENT MTAG ENLIST TOO LATE packet](#)
[TXUSER ENLISTMENT MTAG ENLIST TOO MANY packet](#)
[TXUSER ENLISTMENT MTAG ENLIST TX NOT FOUND packet](#)
[TXUSER ENLISTMENT MTAG ENLISTED packet](#)
[TXUSER ENLISTMENT MTAG PREPAREREQ packet](#)
[TXUSER ENLISTMENT MTAG PREPAREREQDONE packet](#)
[TXUSER ENLISTMENT PREPAREREQDONE RESPONSE enumeration](#)
[TXUSER EXPORT MTAG CREATE packet](#)
[TXUSER EXPORT MTAG CREATE BAD TMADDR packet](#)
[TXUSER EXPORT MTAG CREATE NET TX DISABLED packet](#)
[TXUSER EXPORT MTAG CREATE2 packet](#)
[TXUSER EXPORT MTAG CREATED packet](#)
[TXUSER EXPORT MTAG EXPORT packet](#)
[TXUSER EXPORT MTAG EXPORT LOG FULL packet](#)
[TXUSER EXPORT MTAG EXPORT NO MEM packet](#)
[TXUSER EXPORT MTAG EXPORT TOO LATE packet](#)
[TXUSER EXPORT MTAG EXPORT TOO MANY packet](#)
[TXUSER EXPORT MTAG EXPORT TX NOT FOUND packet](#)
[TXUSER EXPORT MTAG EXPORTED packet](#)
[TXUSER EXTENDEDWHEREABOUTS MTAG GET packet](#)
[TXUSER EXTENDEDWHEREABOUTS MTAG GOT packet](#)
[TXUSER EXTENDEDWHEREABOUTS MTAG NOMEM packet](#)
[TXUSER GETSECURITYFLAGS MTAG FETCHED packet](#)
[TXUSER GETSECURITYFLAGS MTAG GETSECURITYFL AGS packet](#)
[TXUSER GETTXDETAILS MTAG GET packet](#)
[TXUSER GETTXDETAILS MTAG GOTIT packet](#)
[TXUSER GETTXDETAILS MTAG TX NOT FOUND packet](#)
[TXUSER IMPORT MTAG ABORT packet](#)
[TXUSER IMPORT MTAG ABORT TOO LATE packet](#)
[TXUSER IMPORT MTAG IMPORT packet](#)
[TXUSER IMPORT MTAG IMPORT TX NOT FOUND packet](#)
[TXUSER IMPORT MTAG IMPORTED packet](#)
[TXUSER IMPORT MTAG REQUEST COMPLETED packet](#)
[TXUSER IMPORT2 MTAG ABORT packet](#)
[TXUSER IMPORT2 MTAG IMPORT packet](#)
[TXUSER IMPORT2 MTAG IMPORT WITH SET packet](#)
[TXUSER IMPORT2 MTAG SINK ERROR packet](#)
[TXUSER IMPORT2 MTAG SINK IMPORTED packet](#)
[TXUSER PHASE0 MTAG CREATE packet](#)

[TXUSER PHASE0 MTAG CREATE TOO LATE packet](#)
[TXUSER PHASE0 MTAG CREATE TX NOT FOUND packet](#)
[TXUSER PHASE0 MTAG CREATED packet](#)
[TXUSER PHASE0 MTAG PHASE0REQ packet](#)
[TXUSER PHASE0 MTAG PHASE0REQ ABORT packet](#)
[TXUSER PHASE0 MTAG PHASE0REQDONE packet](#)
[TXUSER PHASE0 MTAG UNENLIST packet](#)
[TXUSER REENLIST MTAG REENLIST packet](#)
[TXUSER REENLIST MTAG REENLIST ABORTED packet](#)
[TXUSER REENLIST MTAG REENLIST COMMITTED packet](#)
[TXUSER REENLIST MTAG REENLIST TIMEOUT packet](#)
[TXUSER RESOLVE MTAG ACCESSDENIED packet](#)
[TXUSER RESOLVE MTAG CHILD ABORT packet](#)
[TXUSER RESOLVE MTAG CHILD COMMIT packet](#)
[TXUSER RESOLVE MTAG CHILD NOT PREPARED packet](#)
[TXUSER RESOLVE MTAG FORGET COMMITTED packet](#)
[TXUSER RESOLVE MTAG FORGET TX NOT COMMITTED packet](#)
[TXUSER RESOLVE MTAG REQUEST COMPLETE packet](#)
[TXUSER RESOLVE MTAG TX NOT FOUND packet](#)
[TXUSER RESOURCEMANAGER MTAG CREATE packet](#)
[TXUSER RESOURCEMANAGER MTAG DUPLICATE packet](#)
[TXUSER RESOURCEMANAGER MTAG REENLISTMENTC OMplete packet](#)
[TXUSER RESOURCEMANAGER MTAG REQUEST COMPLETE packet](#)
[TXUSER RESOURCEMANAGERINTERNAL MTAG DUPLICATEDETECTED packet](#)
[TXUSER SETTXTIMEOUT MTAG REQUEST COMPLETE packet](#)
[TXUSER SETTXTIMEOUT MTAG SETTXTIMEOUT packet](#)
[TXUSER SETTXTIMEOUT MTAG TOO LATE packet](#)
[TXUSER SETTXTIMEOUT MTAG TX NOT FOUND packet](#)
[TXUSER STATUS MTAG ABORTED packet](#)
[TXUSER STATUS MTAG COMMITTED packet](#)
[TXUSER STATUS MTAG INDOUBT packet](#)
[TXUSER TRACE MTAG DUMP TRANSACTION packet](#)
[TXUSER TRACE MTAG REQUEST COMPLETE packet](#)
[TXUSER TRACE MTAG REQUEST FAILED packet](#)
[TXUSER TRACE MTAG TX NOT FOUND packet](#)
[TXUSER VOTER MTAG CREATE packet](#)
[TXUSER VOTER MTAG CREATE TOO LATE packet](#)
[TXUSER VOTER MTAG CREATE TX NOT FOUND packet](#)
[TXUSER VOTER MTAG CREATED packet](#)
[TXUSER VOTER MTAG VOTEREQ packet](#)
[TXUSER VOTER MTAG VOTEREQDONE packet](#)
[TXUSER VOTER VOTEREQDONE RESPONSE enumeration](#)

U

[Unilateral abort](#)

V

[Vendor-extensible fields](#)

[Version values - computing](#)

[Versioning](#) ([section 1.7](#), [section 2.2.1](#))

[Versioning - core transaction manager](#)

[Versioning mechanisms](#) ([section 1.7.1](#), [section 2.2.2](#),
[section 2.2.3](#))

[Versioning negotiation mechanisms](#)

[Voting - resource manager](#)

W

[Windows behavior](#)