[MS-COM]:
Component Object Model Plus (COM+) Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
12/18/2006	0.1		MCPP Milestone 2 Initial Availability
03/02/2007	1.0		MCPP Milestone 2
04/03/2007	1.1		Monthly release
05/11/2007	1.2		Monthly release

Date	Revision History	Revision Class	Comments
06/01/2007	1.2.1	Editorial	Revised and edited the technical content.
07/03/2007	1.2.2	Editorial	Revised and edited the technical content.
07/20/2007	1.2.3	Editorial	Revised and edited the technical content.
08/10/2007	1.2.4	Editorial	Revised and edited the technical content.
09/28/2007	1.2.5	Editorial	Revised and edited the technical content.
10/23/2007	1.2.6	Editorial	Revised and edited the technical content.
11/30/2007	1.2.7	Editorial	Revised and edited the technical content.
01/25/2008	1.2.8	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

The Component Object Model Plus (COM+) Protocol consists of a Microsoft proprietary DCOM interface (and DCOM Remote Protocol extensions) used for adding **transactions**, implementing synchronization, managing multiple **object class** configurations, enforcing security, and providing additional functionality and attributes to DCOM-based distributed object applications.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Authentication
Authentication Level
Authentication Service (AS)
Causality Identifier (CID)
Class Factory
Computer Name
Domain
Dynamic Endpoint
Endpoint
Globally Unique Identifier (GUID)
Impersonation Level
Interface Definition Language (IDL)
Microsoft Interface Definition Language (MIDL)
Network Data Representation (NDR)
Object Class
Object RPC (ORPC)
OBJREF
Opnum
Remote Procedure Call (RPC)
RPC Protocol Sequence
RPC Transfer Syntax
RPC Transport
Security Identifier (SID)
Security Provider
Unicode
Universally Unique Identifier (UUID)
Well-Known Endpoint
Windows NT Account Name

The following terms are defined in [MS-DTCO]:

Distributed Transaction

The following terms are specific to this document:

Activity: A synchronization boundary; **ORPC** calls to objects within the boundary are serialized based on their **causality identifiers**.

Activity Identifier: A **GUID** that identifies an activity.

Global Partition: The default, required partition on a machine.

Partition: A container for a specific configuration of a COM+ **object class**.

Partition Identifier: A **GUID** that identifies a partition.

Transaction: A unit of interaction that guarantees the ACID properties— atomicity, consistency, isolation, and durability—as specified by the MSDTC Connection Manager: OleTx Transaction Protocol ([MS-DTCO]).

Transaction Sequence Number (TSN): A positive number that identifies a single **transaction** within a **transaction stream**.

Transaction Stream: An object that supplies a series of transactions, each identified by a monotonically increasing sequence number.

Transaction Stream ID: A **GUID** that identifies a **transaction stream**.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described 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.

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

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", March 2007.

[MS-DTCO] Microsoft Corporation, "MSDTC Connection Manager: OleTx Transaction Protocol Specification", July 2007.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

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

[MS-NRPC] Microsoft Corporation, "Netlogon Remote Protocol Specification", March 2007.

[MS-OAUT] Microsoft Corporation, "OLE Automation Protocol Specification", March 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 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

1.2.2 Informative References

[MSDN-COM] Microsoft Corporation, "Component Object Model", http://msdn2.microsoft.com/en-us/library/aa286559.aspx

[MSDN-DTC] Microsoft Corporation, "Distributed Transaction Coordinator", http://msdn2.microsoft.com/en-us/library/ms684146.aspx

1.3 Protocol Overview (Synopsis)

The COM+ Protocol extends the Distributed Component Object Model (DCOM) Remote Protocol by providing facilities to add transactions, synchronization, multiple object class configurations, security, and other attributes to distributed object applications. The protocol consists of a set of extensions layered on top of the DCOM Remote Protocol. The following diagram shows the layering of the protocol stack.

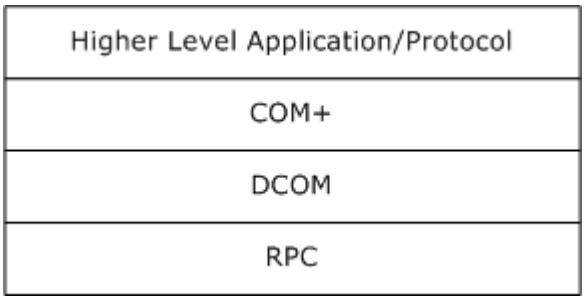


Figure 1: Layering of the protocol stack

1.3.1 Context Properties

The COM+ Protocol operates by passing COM+ specific information in object activations, **ORPC** calls, and as part of marshaled **OBJREFs**, using the context and context property extension mechanisms as specified in [MS-DCOM] section 2.2.1.20.4. A COM+ object is configured using an implementation-specific mechanism to require none, some, or all of the features described in this specification. These features are implemented by creating the object within a context and associating properties with the context. A context is a collection of attributes or context properties that describes an execution environment. When an object in a context with one or more context properties creates or calls other objects on the network, the protocol specifies mechanisms for those context properties to influence the state of the object activations and ORPC calls.

1.3.1.1 Context Properties and Activations

Context properties may flow as part of activation. [MS-DCOM] specifies two ways to accomplish this: the client context and the prototype context passed as part of an **ActivationContextInfoData** structure ([MS-DCOM] section 2.2.1.21.2.5) within an Activation Properties BLOB ([MS-DCOM] section 2.2.1.21).

1.3.1.1.1 Client Context Within Activations

The client context within the Activation Properties BLOB ([MS-DCOM] section 2.2.1.21) represents a set of context properties associated with the client object context, and guides the creation of the server object context.

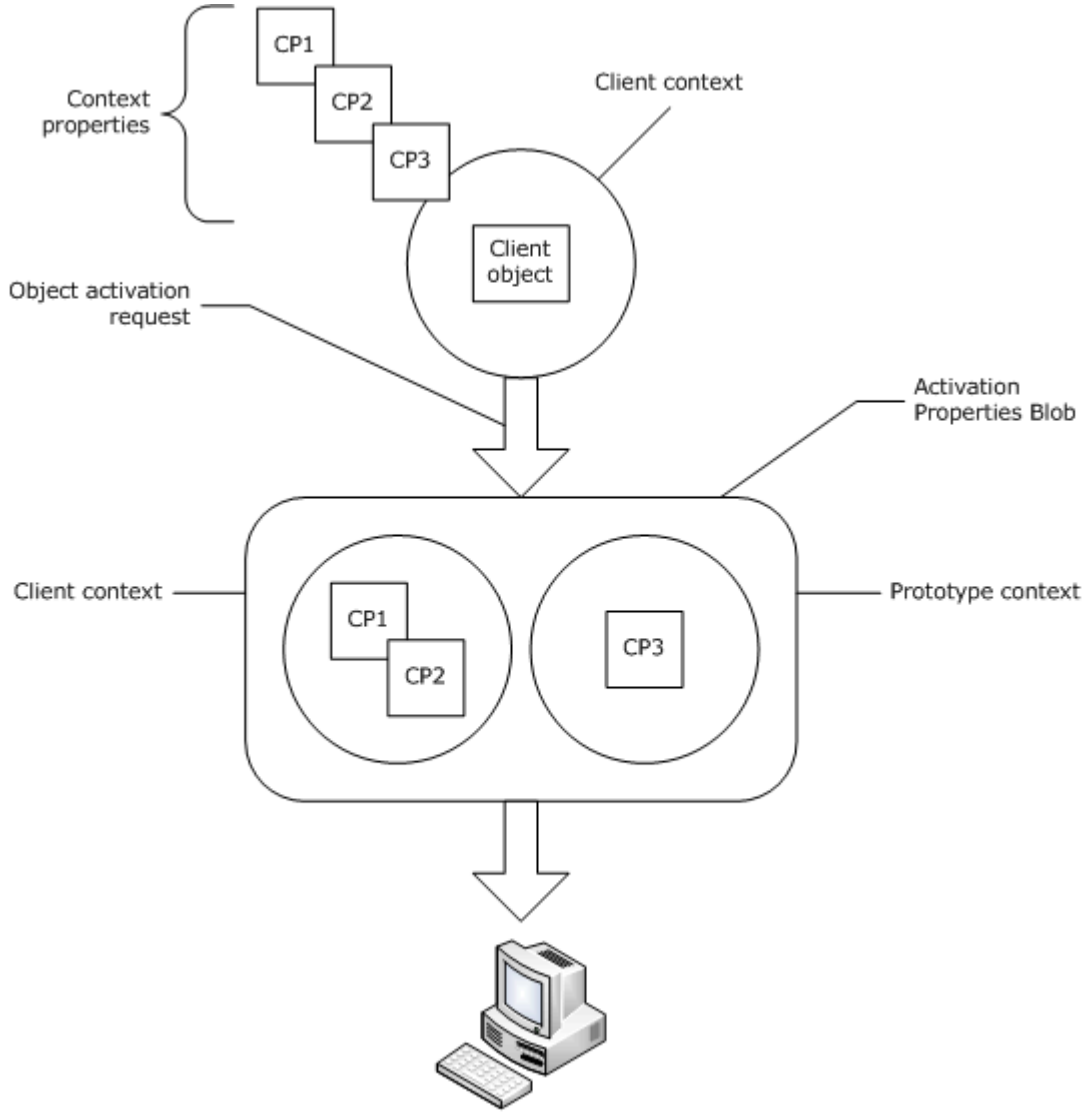
The server may decide to use some, none, or all of the client context properties, depending on the desired result or implementation-specific details.

For example, if the client context contains a transaction context property (section 2.2.2.1), this indicates to the server that the client object is running within a transaction. The server must then decide, in an implementation-specific way, if the server object will run within the same transaction as the client, a new transaction, or no transaction at all.

1.3.1.1.2 Prototype Context Within Activations

The prototype context within the Activation Properties BLOB ([MS-DCOM] section 2.2.1.21) represents the set of context properties of the client that the server must add to its object context. While the client context is merely advisory, the prototype context is not. All the prototype context properties must be present among the context properties of the server object.

1.3.1.1.3 Diagram



A client object sending an activation with context properties to a server.

Figure 2: A client object sending an activation with context properties to a server.

1.3.1.2 Context Properties and Marshaling

When a server marshals a COM+ object running in a context, the server returns an OBJREF_EXTENDED instance ([MS-DCOM] section 2.2.1.17.7). Within an OBJREF_EXTENDED instance, the server can include a representation of its context called an envoy context, consisting of envoy context properties. During unmarshaling, a client may use the envoy context properties to configure and influence future client-side behaviors, either in general or specifically with respect to future communication with the unmarshaled reference.

For example, a marshaled transactional COM+ server object returns a transactional envoy context property in an ORPC call, thereby allowing the client to determine whether the client and the server share the same transaction. If they do not, the client may ignore the transactional envoy context property. If they share the same transaction, the client may send extra information on subsequent ORPC calls to the server, for example, to denote the current **transaction sequence number**, or to send a new transaction if the previous transaction has ended.

1.3.1.2.1 Diagram

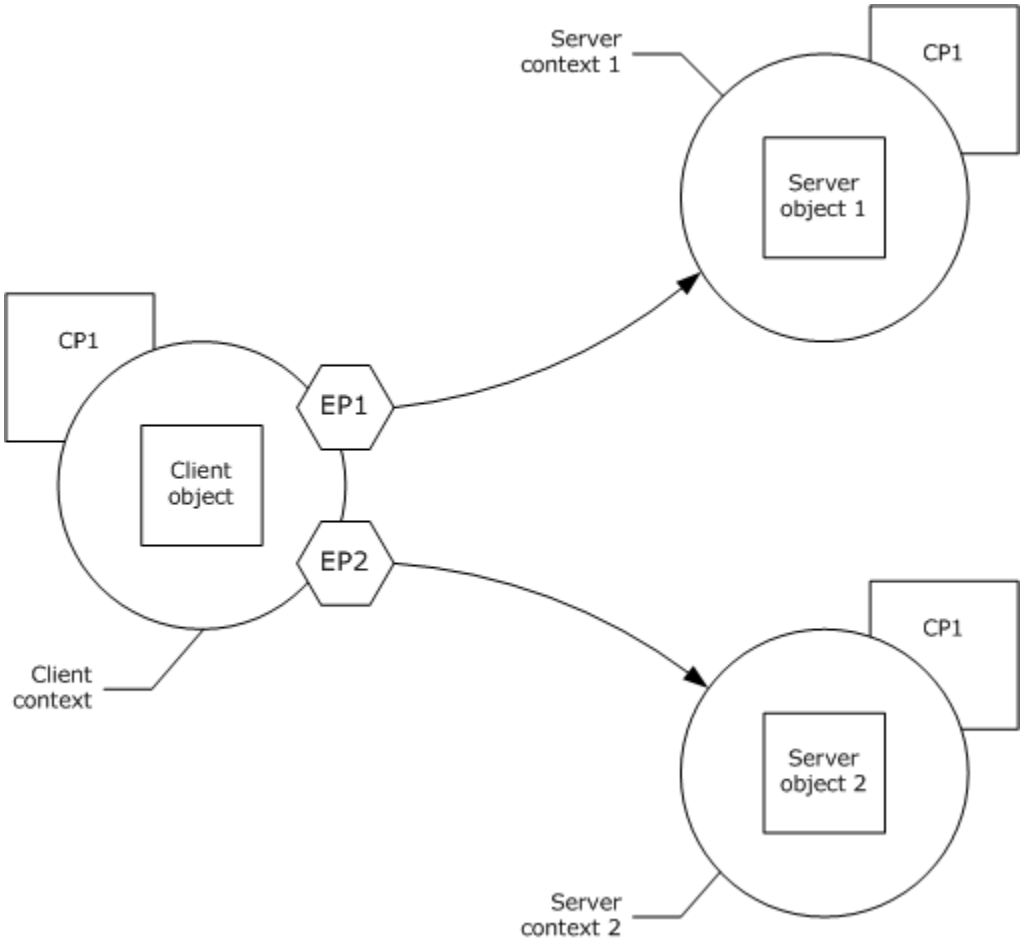
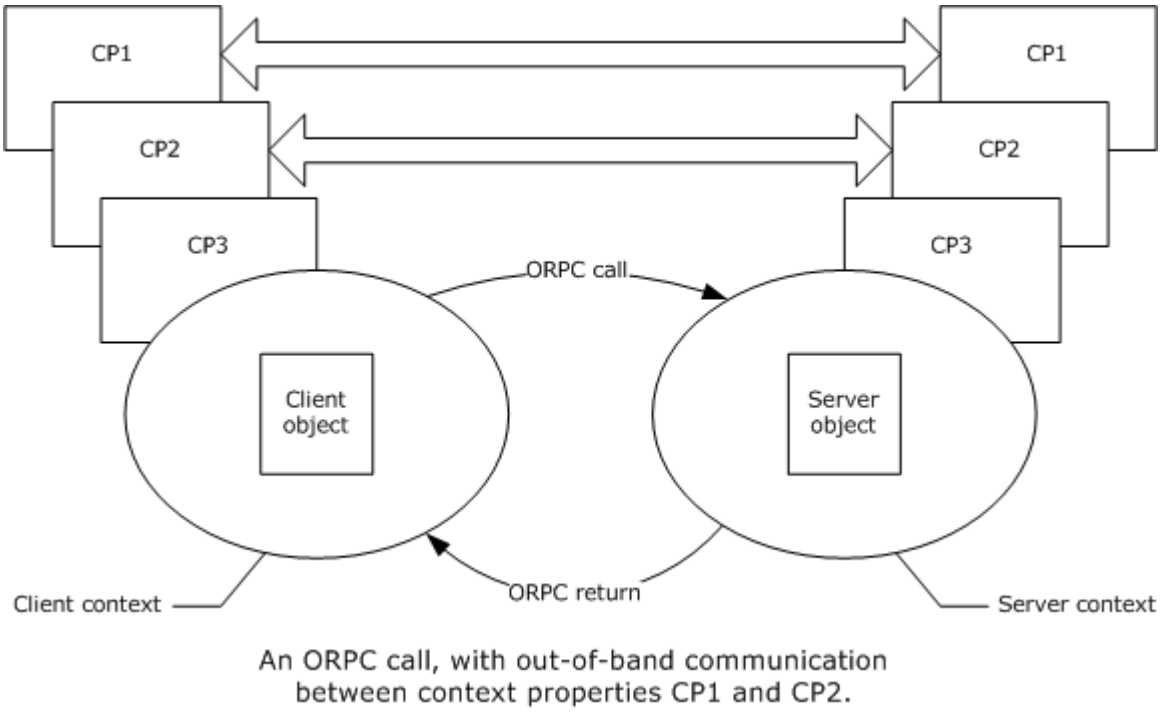


Figure 3: A client object with references to two server objects, each with a reference-specific envoy property (EP1 and EP2) returned from the server during marshaling.

1.3.1.3 Context Properties and ORPC Calls

Context properties may participate in out-of-band communication on ORPC calls. Via the Context ORPC Extension mechanism ([MS-DCOM] section 2.2.1.20.4), a client-side context property and a server-side context property can pass information back and forth on ORPC calls. In some cases, such communication is influenced by an envoy context property returned from the server during the marshaling/unmarshaling process. For example, during an ORPC call a transactional COM+ client object may send extra information about the state of the current transaction to a COM+ server object.

1.3.1.3.1 Diagram



An ORPC call, with out-of-band communication between context properties CP1 and CP2.

Figure 4: An ORPC call with out-of-band communication between context properties

1.3.2 Transactions

The COM+ Protocol is designed to combine the work of collaborating objects under the aegis of a single **distributed transaction**. The COM+ Protocol itself does not define or implement distributed transaction coordination and resource manager facilities; instead, it relies on the MSDTC Connection Manager: OleTx Transaction Protocol ([MS-DTCO]) for these operations, and all references to a "transaction" in this specification should be interpreted to mean a [MS-DTCO] transaction. The COM+ Protocol implements transactional semantics for objects by extending the DCOM Remote Protocol to send transactions and associated information during object activations, in ORPC calls, and within marshaled OBJREF instances.

1.3.2.1 Transaction Stream

A **transaction stream** is an object that supplies a series of transactions, each identified by a monotonically increasing transaction sequence number (TSN). Each transaction stream is uniquely identified by a **GUID** known as a **transaction stream ID**. The TSN is used to synchronize the

transaction participants to the current active transaction. A new transaction in the stream may not be initiated until the previous transaction has completed.

Transaction streams make it possible for sets of distributed objects to collaborate on sequential units of work.

1.3.2.2 Root Transaction Object

The root transaction object is the object for which the initial transaction is created. There can only be one root transaction object within a transaction. The root transaction object has an associated transaction stream, which is responsible for supplying a series of transactions to the root object, as well as to all non-root objects, as required.

1.3.2.3 Non-Root Transaction Object

Non-root transaction objects are objects created by or downstream from the root transaction object, and those that share the root transaction object's transaction. There can be multiple non-root objects within a transaction. A non-root transaction object communicates with the root transaction object and its associated transaction stream to ensure that each ORPC call to the non-root transaction object is always executed using a valid and current transaction.

After a transaction completes, the non-root transaction object retrieves the next transaction either by communicating with the transaction stream, or by receiving it directly as part of an ORPC call from another object running within the transaction.

1.3.2.4 Diagram

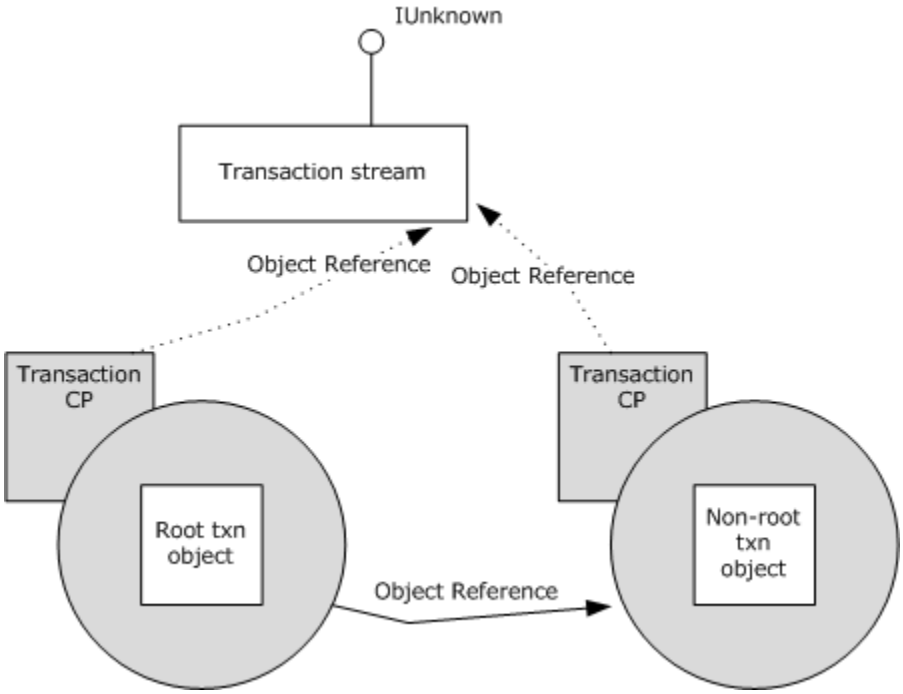


Figure 5: A root and non-root transaction object, each with a transaction context property holding a reference to the transaction stream

1.3.2.5 MS-DTC Transaction Propagation Methods

The MSDTC Connection Manager: OleTx Transaction Protocol specifies two methods for propagating a transaction from one machine to another. For historical reasons, the COM+ Protocol accommodates [MS-DTCO] implementations on client and server machines that support either or both methods. For more information, see section 2.2.6.1 and [MSDN-DTC].

1.3.2.6 Transaction Lifetime

Transactions in the COM+ Protocol are started only by the root transaction object. Only the root transaction object can commit a transaction. A transaction may be canceled by any participant in the transaction.

1.3.3 Activities

An **activity** is a synchronization boundary; ORPC calls to objects within the boundary are serialized based on the DCOM causality of the currently executing ORPC call. An activity is uniquely identified by a GUID known as an **activity ID**. If an object in an activity is currently executing an incoming ORPC call, incoming ORPC calls with different **causality identifiers** (CIDs, as specified in [MS-DCOM] section 2.2.1.6) to other objects within the same activity are blocked for a specified period of time. If the time-out expires before the incoming ORPC call is allowed to enter the activity, the call is rejected and an error is returned to the client.

1.3.4 Security

The protocol offers the capability to send a collection of security identities and other security information along an ORPC call chain; each element in the collection represents a caller in the ORPC call chain. At any point in the call chain, an object may query, in an implementation-specific manner, the following security attributes associated with each upstream caller:

- The caller's identity (specified by a security identifier (SID) or Windows NT account name).
- The authentication service of the call.
- The authentication level of the call.
- The impersonation level of the call.

In addition, an object in the call chain may also query the minimum **authentication level** used across the entire call chain.

The protocol uses the security context property to send security information in ORPC calls as described in section 1.3.1.3. When an object is marshaled, the protocol uses the security envoy property (section 2.2.4.2) as described in section 1.3.1.2 to send information about the **domain** and computer of the object. The protocol uses this information to translate **SIDs** to **Windows NT account names** when sending the security identity of the caller in cross-computer and cross-domain ORPC calls.

1.3.5 User-Defined Properties

User-defined properties are name/value pairs that are part of an object's context. These properties are supplied and consumed by higher-level protocols or applications. The COM+ Protocol supports the string value type and the OBJREF value type. The protocol sends these properties as part of both the client and prototype contexts during object activations.

1.3.6 Partitions

Partitions are used to support the side-by-side installation of multiple configurations of a COM+ object class. Each partition is uniquely identified by a GUID known as the **partition ID**. An object class may have several versions of its configuration installed on a server, one per partition. A partition must contain at most one version of an object class.

Every machine has at minimum one partition, the **global partition**, which contains the default configuration for every object class on the machine. The global partition serves as the default partition when no criteria, such as a client-specified partition, exist to choose any other partition.

The partition of an object class is determined during the object activation request; it may be chosen automatically by an implementation on behalf of the activating client, or the activating client may specify a partition ID as part of the activation request.

For historical reasons, the client's partition information is not sent across the network in the form of a context property; instead, it is sent as part of the base DCOM protocol. See the **guidPartition** field of the **SpecialPropertiesData** structure, as specified in **[MS-DCOM]** section 2.2.1.21.2.2.

1.4 Relationship to Other Protocols

This protocol is built on top of the DCOM Remote Protocol ([MS-DCOM]). This protocol requires the MSDTC Connection Manager: OleTx Transaction Protocol ([MS-DTCO]) for implementing the transactional features described in this specification.

1.5 Prerequisites/Preconditions

This protocol requires that both client and server possess implementations of the DCOM Remote Protocol. If the transactional features of this protocol are to be used, both client and server must possess implementations of the MSDTC Connection Manager: OleTx Transaction Protocol.

1.6 Applicability Statement

This protocol is useful and appropriate when a distributed, object-based architecture with transactions, synchronization, security, and side-by-side installation of multiple configurations of an object class is required.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- Supported Transports: The protocol MUST use any supported DCOM transport.
- Security and Authentication Methods: The protocol SHOULD use the underlying security and authentication mechanisms provided by RPC and DCOM.
- Capability Negotiation: The protocol MUST perform explicit capability negotiation, as follows.
  - By use of the **COMVERSION** structure ([MS-DCOM] section 2.2.1.11), which is passed between client and server; clients and servers associate specific version numbers with specific capabilities and behaviors.
  - By the use of version numbers embedded within marshaled buffers.

1.8 Vendor-Extensible Fields

This protocol uses **HRESULT**s as specified in [MS-ERREF] section 2.1. Vendors can define their own **HRESULT** values, provided that the C bit (0x20000000) is set for each vendor-specific value.

1.9 Standards Assignments

The following is a table of well-known GUIDs in the COM+ Protocol.

Name	GUID	Purpose	Section
guidTransactionProperty	{ecabaeb1-7f19-11d2-978e-0000f8757e2a}	Transaction context property identifier Transaction ORPC extension identifier Transaction envoy property identifier	2.2.2.1 2.2.3.1 2.2.4.1
guidActivityProperty	{ecabaeb4-7f19-11d2-978e-0000f8757e2a}	Activity context propertyPidentifier	2.2.2.2
guidSecurityProperty	{ecabaeb8-7f19-11d2-978e-0000f8757e2a}	Security envoy property identifier Security ORPC extension identifier	2.2.4.2 2.2.3.2.4
guidUserPropertiesProperty	{ecabaeb6-7f19-11d2-978e-0000f8757e2a}	User-defined context property identifier	2.2.2.3
CLSID_CFW	{ecabafc0-7f19-11d2-978e-0000f8757e2a}	OBJREF_CUSTOM unmarshaler **CLSID** for Class factory wrapper	2.2.5
CLSID_UserContextProperty	{ecabafb3-7f19-11d2-978e-0000f8757e2a}	OBJREF_CUSTOM unmarshaler **CLSID** for user-defined context property.	2.2.2.3
CLSID_TransactionUnmarshal	{ecabafac-7f19-11d2-978e-0000f8757e2a}	OBJREF_CUSTOM unmarshaler **CLSID** for transaction context property.	2.2.2.1
CLSID_ActivityUnmarshal	{ecabafaa-7f19-11d2-978e-0000f8757e2a}	OBJREF_CUSTOM unmarshaler **CLSID** for activity context property.	2.2.2.2
CLSID_SecurityEnvoy	{ ecabafab-7f19-11d2-978e-0000f8757e2a}	The unmarshaling **CLSID** for the security envoy property.	2.2.4.2
CLSID_TransactionEnvoy	{ ecabafad-7f19-11d2-978e-0000f8757e2a}	The unmarshaling **CLSID** for the transaction envoy property.	2.2.4.1

Parameter	Value
RPC Interface UUID	97199110-DB2E-11d1-A251-0000F805CA53

2 Messages

All structures are defined in the **IDL** syntax and are marshaled as per [C706] part 3. The IDL is specified in Appendix A.

Field types in packet diagrams are specified by the packet diagram and the field descriptions. All integer-based fields in packet diagrams are marshaled using little-endian byte ordering unless otherwise specified.

All extra padding bytes MUST be zero, unless otherwise specified, and they MUST be ignored on receipt.

2.1 Transport

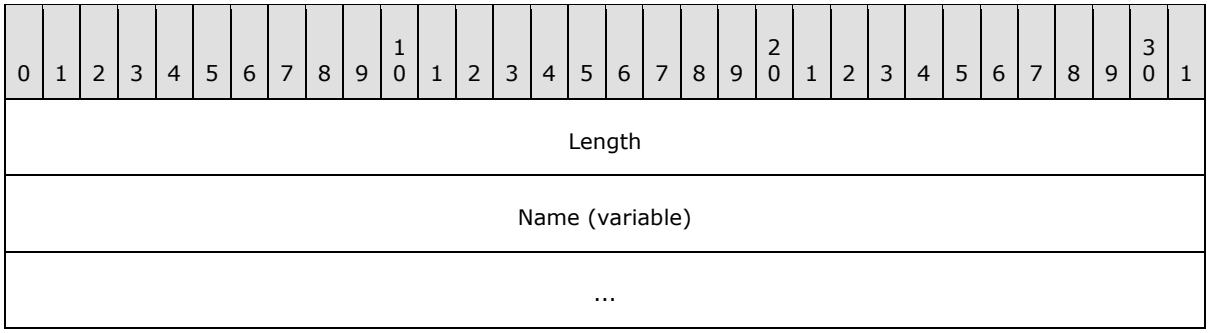
This protocol uses **RPC dynamic endpoints** as specified in [C706] part 4.

2.2 Common Data Types

In addition to RPC base types and definitions specified in [C706] and [MS-DTYP], additional data types are specified in this section.

2.2.1 LengthPrefixedName

The LengthPrefixedName type specifies an array of **Unicode** characters prefixed by the array length, in characters.



Length (4 bytes): An unsigned long that MUST contain the number of Unicode characters in **Name**, and MUST NOT be zero.

Name (variable): A Unicode string; the string SHOULD NOT end in a NULL terminator.

2.2.2 Activation Context Properties

Activation context properties are included as part of the client and/or prototype contexts in a DCOM Activation Properties BLOB ([MS-DCOM] section 2.2.1.21).

The following table shows which context properties should be located within either the client or prototype contexts.

Context property	In client or prototype context?
Transaction (section 2.2.2.1)	If present, this property MUST be in client context only.

Context property	In client or prototype context?
Activity (section 2.2.2.2)	If present, this property MUST be in client context only.
User-defined (section 2.2.2.3)	If present, this property MUST be in both client and prototype contexts.

The user context property, if present, MUST be sent within both the client and prototype contexts, and both copies MUST be identical.

2.2.2.1 Transaction Context Property

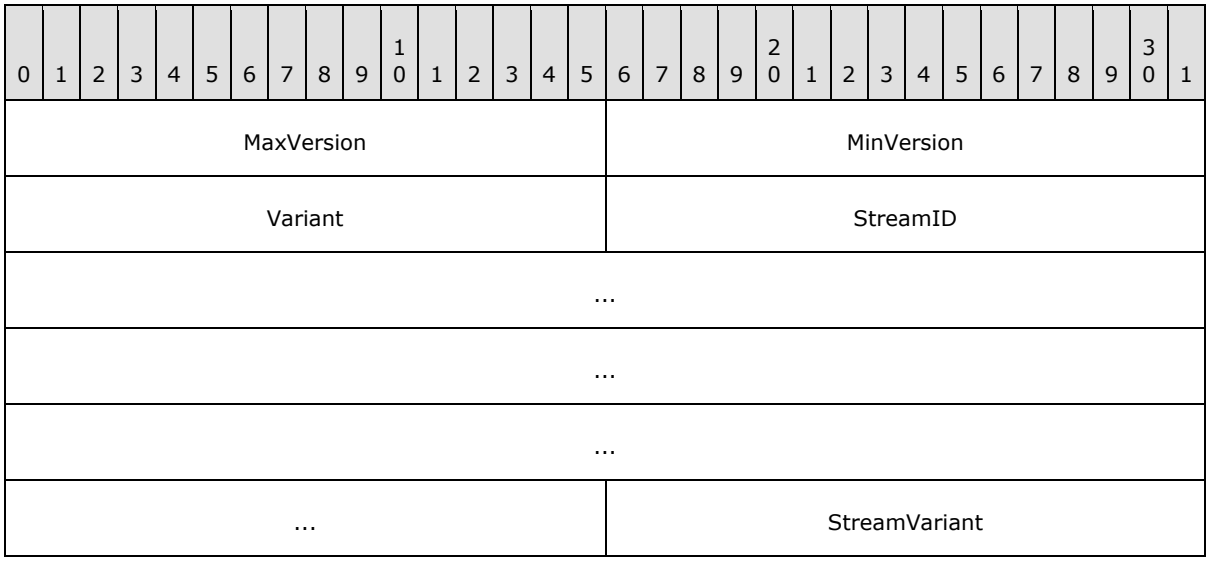
To indicate to the server that the client is running within a transaction, the client MUST include a transaction context property as part of the client context in an object activation request.

The **policyId** field of the PROPMARSHALHEADER instance ([MS-DCOM] section 2.2.1.19.1) for the transaction context property MUST be set to guidTransactionProperty, as specified in section 1.9. The **CLSID** field of the PROPMARSHALHEADER instance ([MS-DCOM] section 2.2.1.19.1) for the transaction context property MUST be set to GUID_NULL. The transaction context property MUST be marshaled using the OBJREF_CUSTOM format ([MS-DCOM] section 2.2.1.17.6), and the **CLSID** field of the OBJREF_CUSTOM instance MUST be set to CLSID_TransactionUnmarshal, as specified in section 1.9.

The format of the OBJREF_CUSTOM.pObjectData buffer for CLSID_TransactionUnmarshal MUST be specified as follows.

2.2.2.1.1 TransactionContextPropertyHeader

The TransactionContextPropertyHeader structure is the common header for all variants of the Transaction Context Property.



MaxVersion (2 bytes): The major version of this marshaled format. MUST be set to 0x0001 or 0x0002. A value of 0x0002 indicates that an **IsolationLevel** field is present at the end of the message (see sections 2.2.2.1.2 and 2.2.2.1.3); a value of 0x0001 indicates that no **IsolationLevel** is present.

MinVersion (2 bytes): The minor version of this marshaled format. MUST be set to 0x0001.

Variant (2 bytes): This MUST be set to either 0x0000 or 0x0002, and MUST be ignored by the server on receipt.

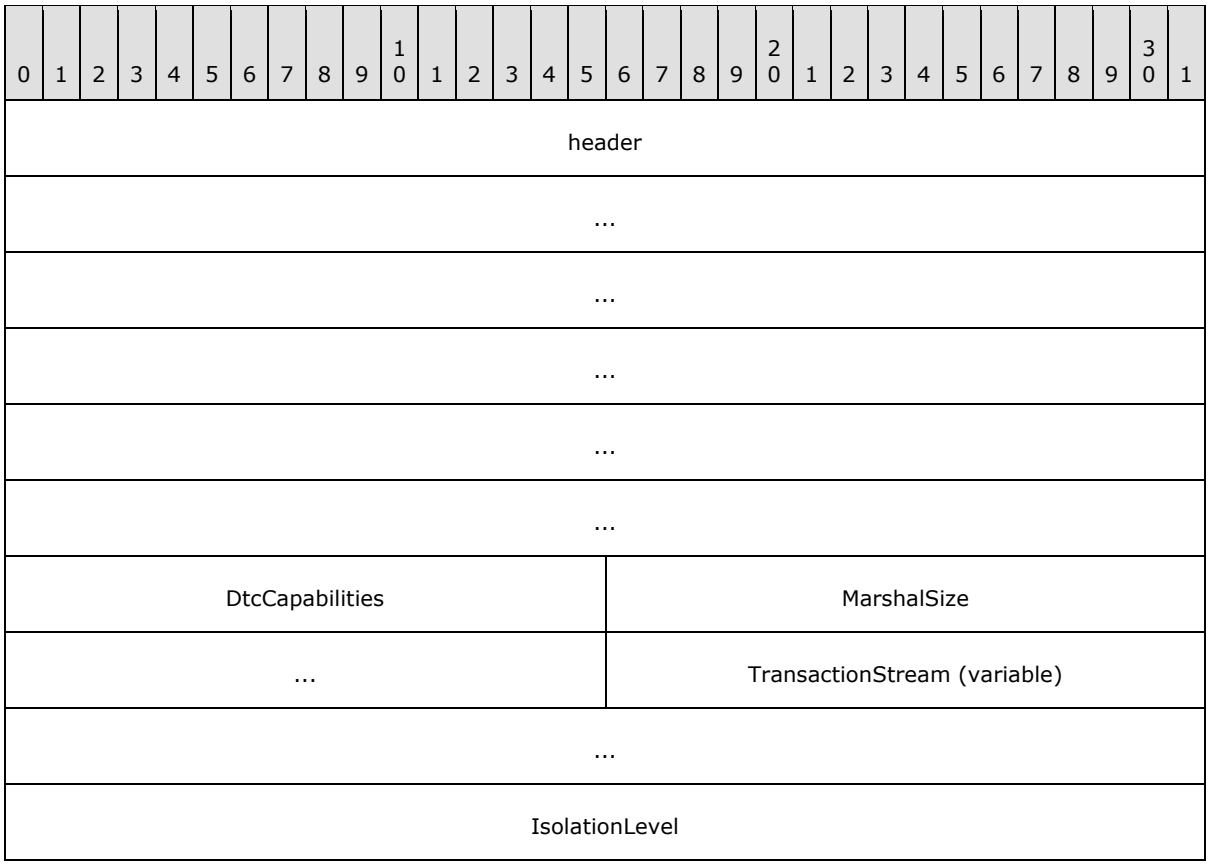
StreamID (16 bytes): A GUID identifying the controlling transaction stream.

StreamVariant (2 bytes): A value identifying the larger structure that contains the TransactionContextPropertyHeader. It MUST be set to one of the following values:

Value	Meaning
StreamVariant 0x0001	The TransactionContextPropertyHeader structure MUST be contained as part of a TransactionStream (section 2.2.2.1.2) structure.
TransactionVariant 0x0002	The TransactionContextPropertyHeader structure MUST be contained as part of a TransactionBuffer (section 2.2.2.1.3) structure.

2.2.2.1.2 TransactionStream

The TransactionStream structure is used when the client passes a reference to the client's **ITransactionStream** interface and conveys information about the capabilities of the [MS-DTCO] implementation on the client.



header (24 bytes): A TransactionContextPropertyHeader; the **StreamVariant** field of the structure MUST be set to 0x0001.

DtcCapabilities (2 bytes): This MUST be set to a bitwise OR of one or more of the values defined in section 2.2.6.1.

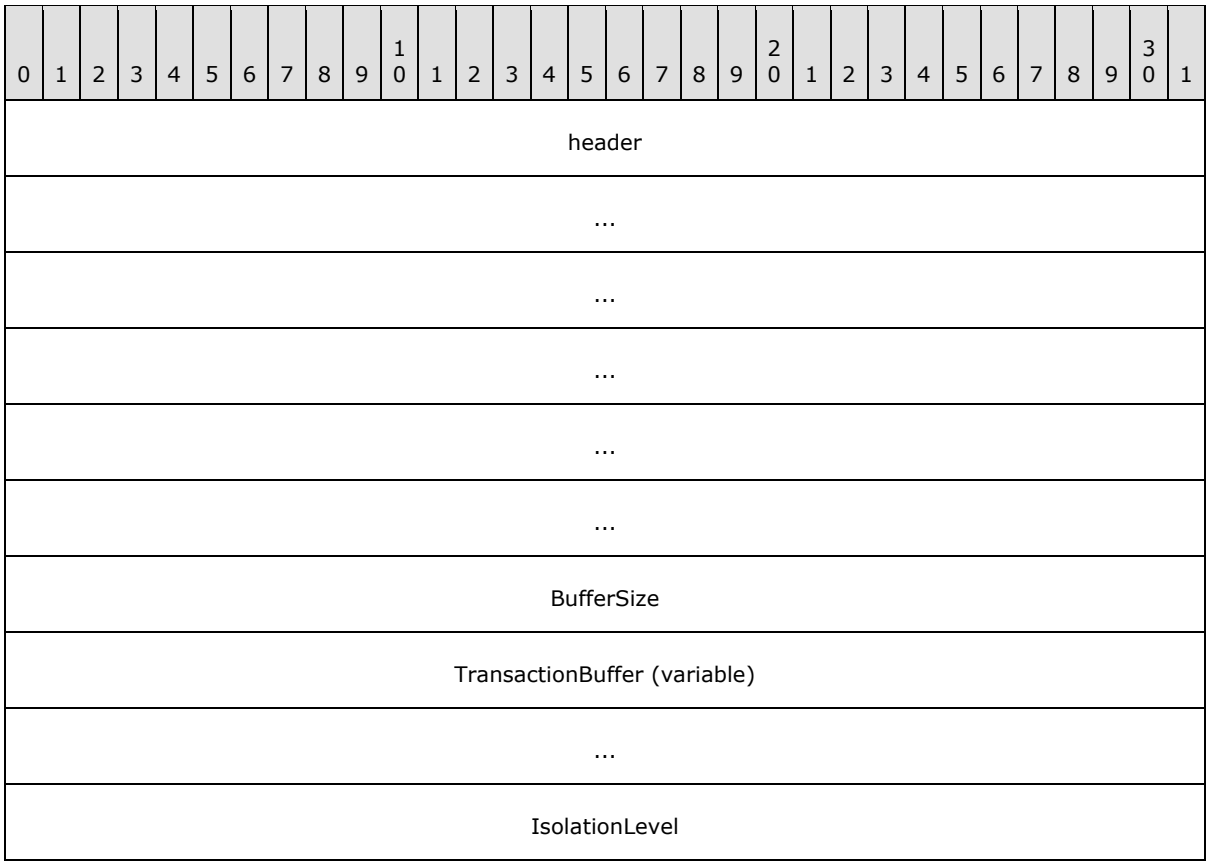
MarshalSize (4 bytes): The (unsigned) size in bytes of **TransactionStream**.

TransactionStream (variable): An OBJREF instance that MUST contain a marshaled **ITransactionStream** interface instance.

IsolationLevel (4 bytes): This field MUST be present if the **MaxVersion** field of header is 0x0002; otherwise, this field MUST NOT be present. If it is present, it MUST contain one of the values defined in section 2.2.6.2.

2.2.2.1.3 TransactionBuffer

The TransactionBuffer structure is used when the client passes the currently active transaction to the server.



header (24 bytes): A TransactionContextPropertyHeader structure. The **StreamVariant** field of the structure MUST be set to 0x0002.

BufferSize (4 bytes): An unsigned long that MUST specify the size, in bytes, of **TransactionBuffer**.

TransactionBuffer (variable): An array of bytes that MUST contain a Propagation_Token structure as specified in [MS-DTCO].

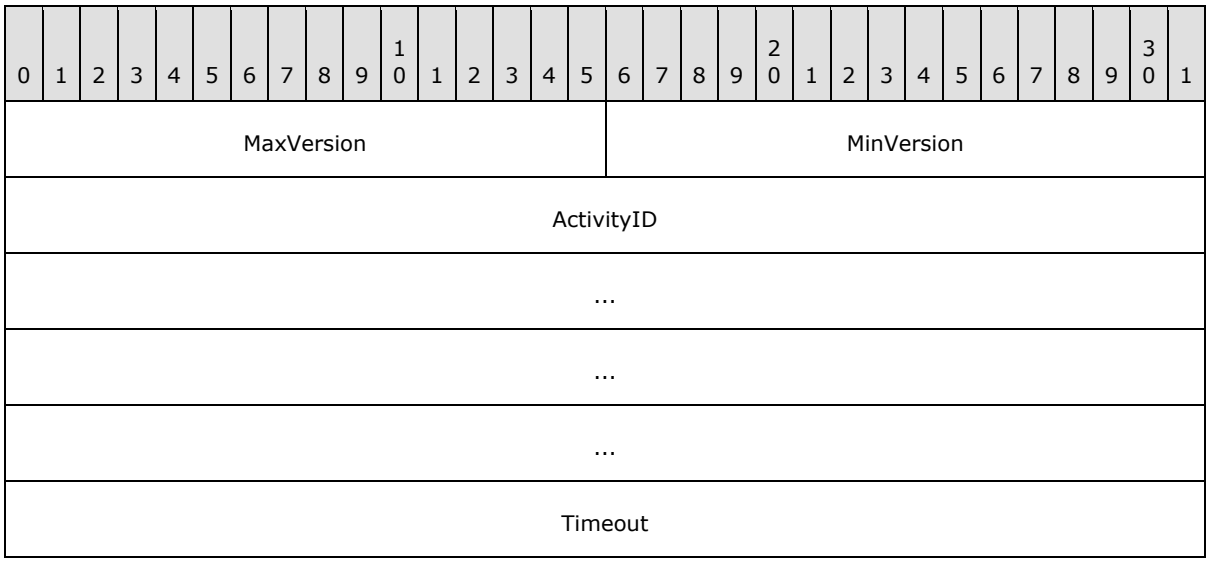
IsolationLevel (4 bytes): This field MUST be present if the **MaxVersion** field of header is 0x0002; otherwise, this field MUST NOT be present. If it is present, it MUST contain one of the values defined in section 2.2.6.2.

2.2.2.2 Activity Context Property

To indicate to the server that the client is running within an activity, the client MUST include an activity context property as part of the client context in an object activation request.

The **policyId** field of the PROPMARSHALHEADER instance ([MS-DCOM] section 2.2.1.19.1) for the activity context property MUST be set to guidActivityProperty, as specified in section 1.9. The **CLSID** field of the PROPMARSHALHEADER instance ([MS-DCOM] section 2.2.1.19.1) for the activity context property MUST be set to GUID_NULL. The activity context property MUST be marshaled using the OBJREF_CUSTOM format ([MS-DCOM] section 2.2.1.17.6), and the **CLSID** field of the OBJREF_CUSTOM instance MUST be set to CLSID_ActivityUnmarshal, as specified in section 1.9.

The format of the OBJREF_CUSTOM.pObjectData buffer for CLSID_ActivityUnmarshal MUST be specified as follows:



MaxVersion (2 bytes): The major version number for this activity context property format. This field MUST be set to 0x0001.

MinVersion (2 bytes): The minor version number for this activity context property format. This field MUST be set to 0x0001.

ActivityID (16 bytes): A GUID that MUST specify the activity ID.

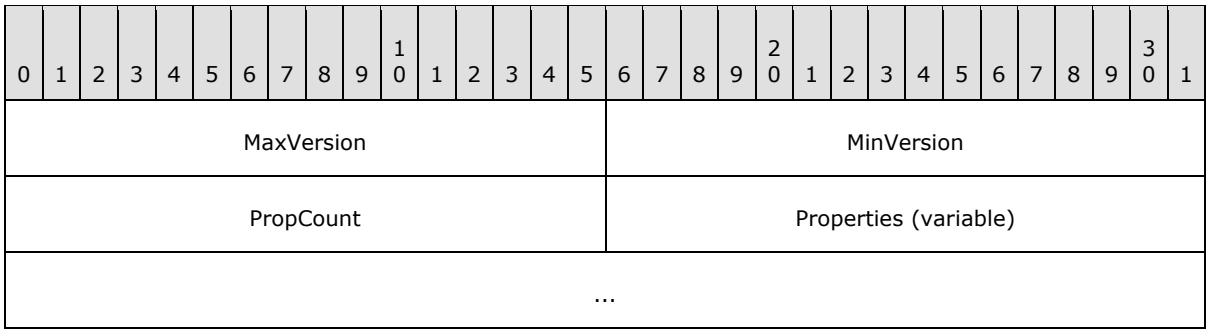
Timeout (4 bytes): An unsigned long that MUST specify the activity time-out in milliseconds. A value of 0xFFFFFFFF MUST be interpreted to specify an INFINITE time-out.

2.2.2.3 User-Defined Context Property

The user-defined context property, if present, MUST be included as part of both the client and prototype contexts during activation requests. This context property contains a logical set of name/value pairs.

The **policyId** field of the PROPMARSHALHEADER instance ([MS-DCOM] section 2.2.1.19.1) for the user-defined context property MUST be set to guidUserPropertiesProperty, as specified in section 1.9). The **CLSID** field of the PROPMARSHALHEADER instance ([MS-DCOM] section 2.2.1.19.1) for the activity context property MUST be set to GUID_NULL. The user-defined context property MUST be marshaled using the OBJREF_CUSTOM format ([MS-DCOM] section 2.2.1.17.6), and the **CLSID** field of the OBJREF_CUSTOM instance MUST be set to CLSID_UserContextProperty, as specified in section 1.9.

The format of the OBJREF_CUSTOM.pObjectData buffer for CLSID_UserContextProperty MUST be specified as follows:



MaxVersion (2 bytes): The major version number for this UserProperty (section 2.2.2.3.1) format. This field MUST be set to 0x0001.

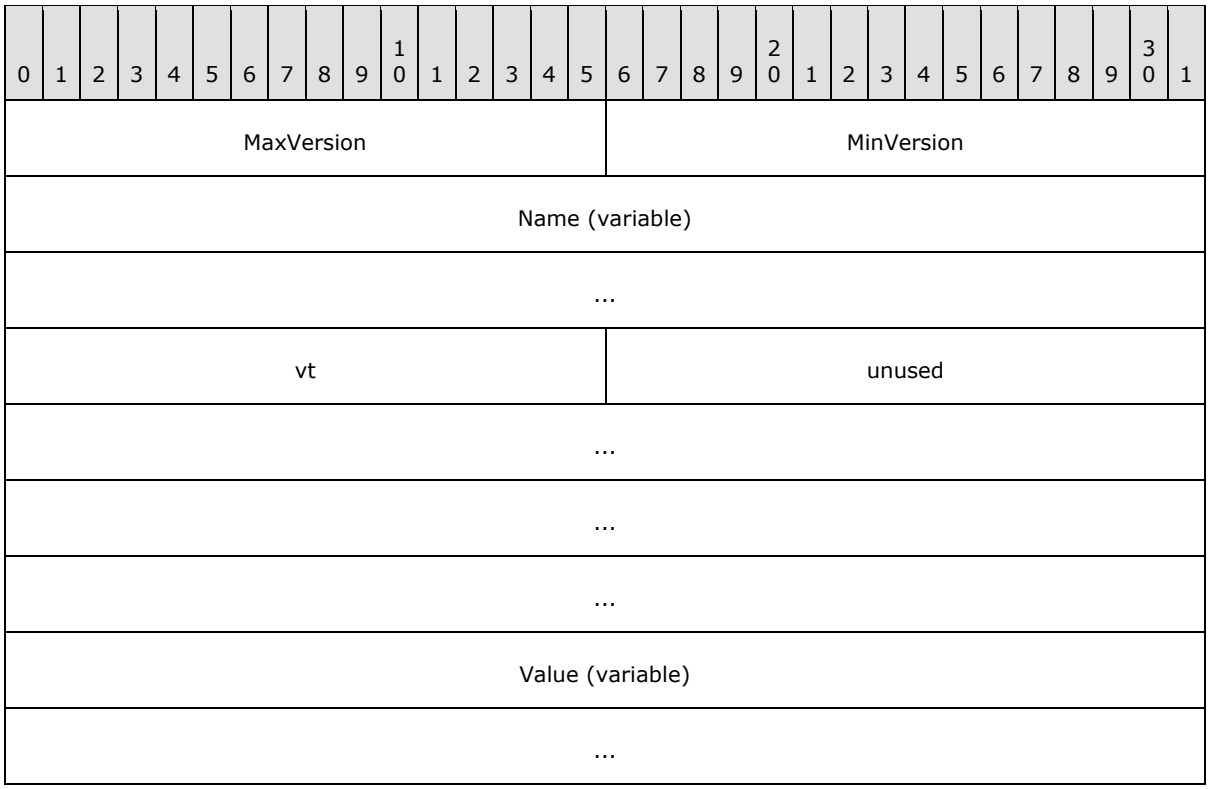
MinVersion (2 bytes): The minor version number for this UserProperty format. This field MUST be set to 0x0001.

PropCount (2 bytes): An unsigned short that MUST specify the number of elements in the **Properties** array.

Properties (variable): An array of UserProperty structures.

2.2.2.3.1 UserProperty

The UserProperty structure is used to define a single name/value pair.



MaxVersion (2 bytes): The major version number for this UserProperty format; this field MUST be set to 0x0001.

MinVersion (2 bytes): The minor version number for this UserProperty format; this field MUST be set to 0x0001.

Name (variable): A LengthPrefixedName (section 2.2.1) containing the name of the UserProperty.

vt (2 bytes): The type of data contained in **Value**. It MUST be set to one of the following values:

vt value	Meaning
0x0008	A LengthPrefixedName (section 2.2.1).
0x0009	An OBJREF ([MS-DCOM] section 2.2.1.17) with an **iid** field that MUST be set to IID_IUnknown ([MS-DCOM] section 1.9).
0x000D	An OBJREF ([MS-DCOM] section 2.2.1.17) with an **iid** field that MUST be set to IID_IDispatch ([MS-OAUT] section 1.9).

unused (14 bytes): SHOULD be set to zero, and MUST be ignored upon receipt.<1>

Value (variable): MUST contain the data for this name/value pair, as specified by the **vt** field.

2.2.3 Context ORPC Extensions

Context ORPC extensions are specified in [MS-DCOM] section 2.2.1.20.4. These extension formats are passed as out-of-band data on ORPC calls. Each individual extension is identified by a "**policyID**" of its corresponding EntryHeader ([MS-DCOM] section 2.2.1.20.5).

2.2.3.1 Transaction ORPC Extensions

These extensions are used to coordinate the state of a transaction in use by both the client and the server.

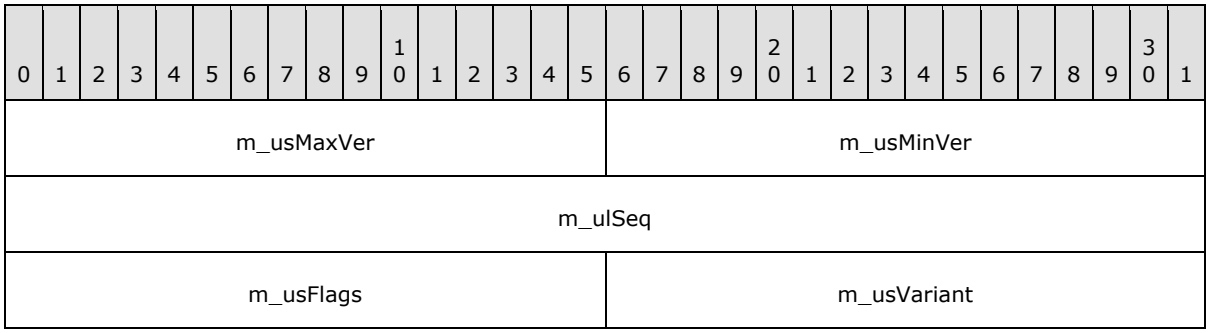
The **policyID** field of the EntryHeader for this extension MUST be set to guidTransactionProperty (section 1.9).

2.2.3.1.1 Transaction ORPC Call Extensions

These extensions are sent by a client in order to inform the server of the current transaction state, and to request that other transaction-related data be returned by the server within the same call.

2.2.3.1.1.1 TransactionPropCallHeader

The TransactionPropCallHeader structure is used to pass the TSN of the current transaction to the server.



m_usMaxVer (2 bytes): The major version number for this TransactionPropCallHeader format. This field MUST be set to 0x0001.

m_usMinVer (2 bytes): The minor version number for this TransactionPropCallHeader format; this field MUST be set to 0x0001.

m_ulSeq (4 bytes): The sequence number of the current transaction.

m_usFlags (2 bytes): This MUST contain one of the following values:

Value	Meaning
0x0000	A request that the server MUST return a TransactionPropRetHeader structure with the **m_usVariant** field set to TransactionPropCall_None, as specified in section 2.2.3.1.2.1.
TransactionPropCallFlag_NeedWhereabouts 0x0001	A request that the server MUST return a TransactionPropRetHeader structure with the **m_usVariant** field set to

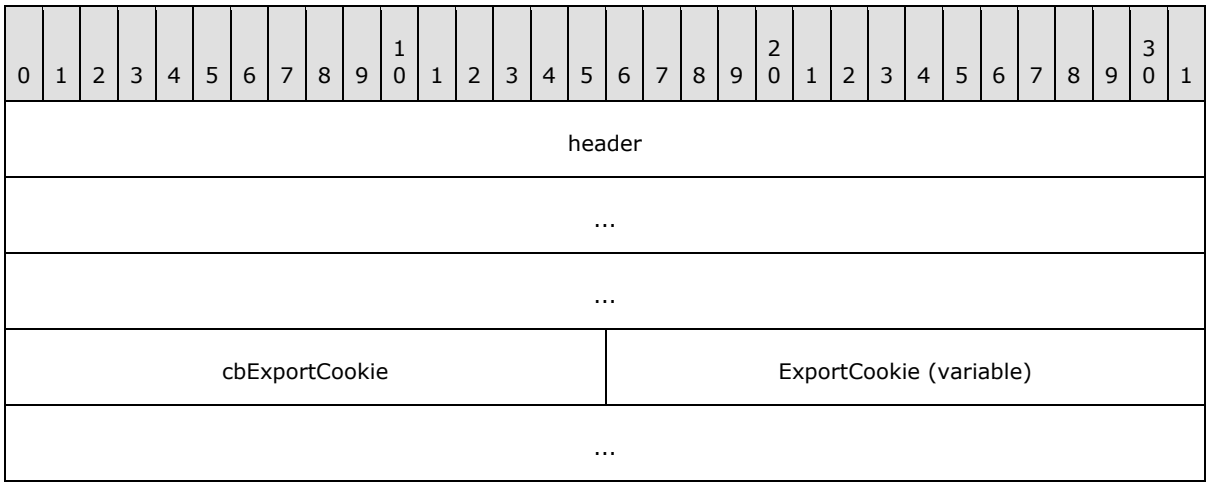
Value	Meaning
	TransactionPropRet_Whereabouts, as specified in section 2.2.3.1.2.1.

m_usVariant (2 bytes): This MUST contain one of the following values:

Value	Meaning
TransactionPropCall_None 0x0001	The TransactionPropCallHeader structure MUST NOT be contained within any larger structures.
TransactionPropCall_ExportCookie 0x0002	The TransactionPropCallHeader structure MUST be contained as part of the TransactionPropCallExportCookie (section 2.2.3.1.1.2) structure.
TransactionPropCall_TransmitterBuffer 0x0003	The TransactionPropCallHeader structure MUST be contained as part of the TransactionPropCallTransmitterBuffer (section 2.2.3.1.1.3) structure.

2.2.3.1.1.2 TransactionPropCallExportCookie

The TransactionPropCallExportCookie structure is used to send the currently active transaction to the server, using the STxInfo format. For more details, see [MS-DTCO].



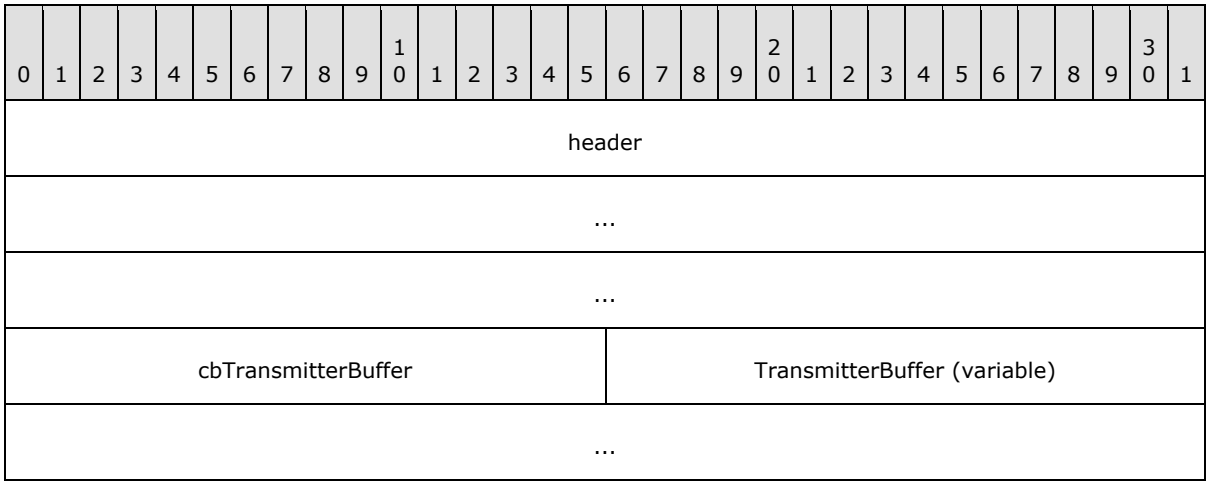
header (12 bytes): A TransactionPropCallHeader. The **m_usVariant** field of the structure MUST be set to TransactionPropCall_ExportCookie (0x0002).

cbExportCookie (2 bytes): An unsigned short that MUST specify the number of bytes contained in **ExportCookie**.

ExportCookie (variable): An array of bytes that MUST contain the STxInfo buffer. For more details, see [MS-DTCO].

2.2.3.1.1.3 TransactionPropCallTransmitterBuffer

The TransactionPropCallTransmitterBuffer structure is used to send the currently active transaction to the server, using the Propagation Token format; see [MS-DTCO] section 2.2.5.4 for more details.



header (12 bytes): A TransactionPropCallHeader. The **m_usVariant** field of the structure MUST be set to TransactionPropCall_TransmitterBuffer (0x0003).

cbTransmitterBuffer (2 bytes): An unsigned short that MUST specify the number of bytes contained in **TransmitterBuffer**.

TransmitterBuffer (variable): An array of bytes that MUST contain a Propagation Token buffer; see [MS-DTCO] section 2.2.5.4 for more details.

2.2.3.1.2 Transaction ORPC Return Extensions

These extensions are returned in the ORPC response by a server in response to one of the call extensions specified in section 2.2.3.1.

The **policyID** field of the EntryHeader for these extensions MUST be set to guidTransactionProperty (section 1.9).

2.2.3.1.2.1 TransactionPropRetHeader

The server uses the TransactionPropRetHeader structure to communicate transaction status, and optionally to return additional data that may advise the client to cancel the current transaction or to stop sending further information about it.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
m_usMaxVer																m_usMinVer															
m_usFlags																m_usVariant															

m_usMaxVer (2 bytes): The major version number for this TransactionPropRetHeader format; this field MUST be set to 0x0001.

m_usMinVer (2 bytes): The minor version number for this TransactionPropRetHeader format; this field MUST be set to 0x0001.

m_usFlags (2 bytes): This MUST contain zero or more of the following flags:

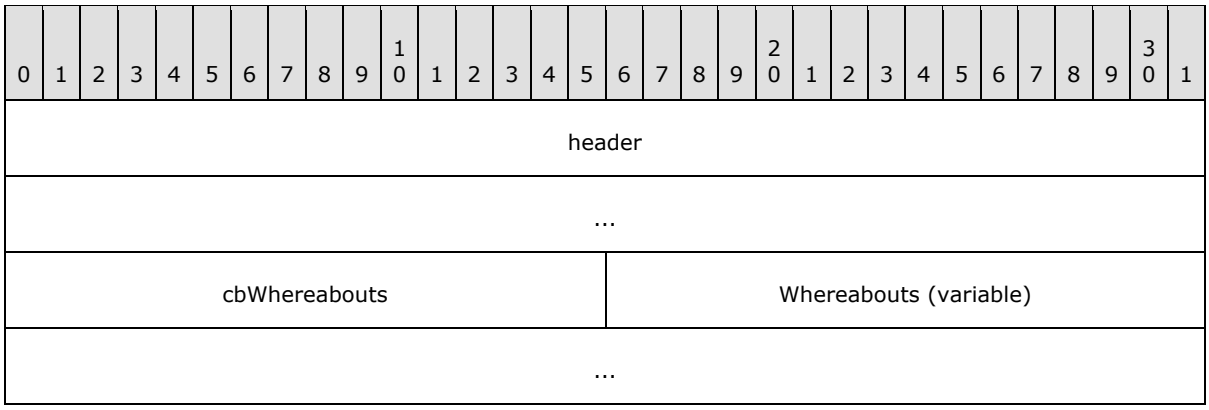
Value	Meaning
TransactionPropRetFlag_Abort 0x0001	The client MUST cancel the transaction.
TransactionPropRetFlag_DontSend 0x0002	The client SHOULD NOT send the currently active transaction (for example, either as a TransactionPropCallExportCookie (section 2.2.3.1.1.2) or as a TransactionPropCallTransmitterBuffer (section 2.2.3.1.1.3)) to the server again for the life of the transaction.

m_usVariant (2 bytes): This MUST be one of the following values:

Value	Meaning
TransactionPropCall_None 0x0000	The TransactionPropRetHeader structure MUST NOT be contained within any larger structures.
TransactionPropRet_Whereabouts 0x0001	The TransactionPropRetHeader structure MUST be contained as part of TransactionPropRetWhereabouts (section 2.2.3.1.2.2).

2.2.3.1.2.2 TransactionPropRetWhereabouts

The TransactionPropRetWhereabouts structure is used by the server to return additional data and to communicate transaction status to the client.



header (8 bytes): A TransactionPropRetHeader (section 2.2.3.1.2.1). The **m_usVariant** field of the structure MUST be set to TransactionPropRet_Whereabouts (0x1).

cbWhereabouts (2 bytes): An unsigned short that MUST specify the number of bytes contained in **Whereabouts**.

Whereabouts (variable): An array of bytes that MUST contain an **SWhereabouts** structure. For more information, see [MS-DTCO].

2.2.3.2 Security ORPC Extension

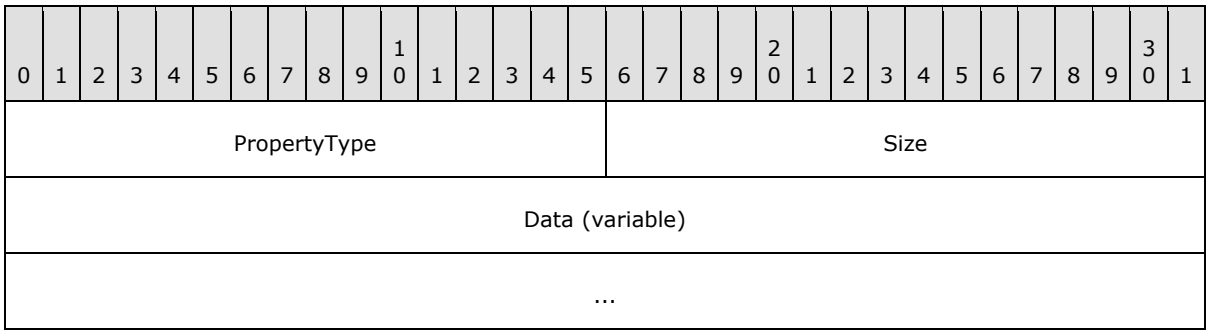
The Security ORPC Extension sends COM+ security information as out-of-band data on ORPC calls between two COM+ objects. The security information provides a record of the chain of caller identities and other security attributes within a series of ORPC calls.

The Security ORPC Extension structure MUST contain an array of Security Property Collection (section 2.2.3.2.3) structures. Each Security Property Collection structure in turn MUST contain an array of Security Property (section 2.2.3.2.1) structures. Each Security Property structure MUST specify a Security Property Type (section 2.2.3.2.1.1).

The **policyID** field of the EntryHeader ([MS-DCOM] section 2.2.1.20.5) of the Security ORPC Extension MUST be set to guidSecurityProperty (section 1.9).

2.2.3.2.1 Security Property

The Security Property structure specifies a security property sent by the security ORPC extension.



PropertyType (2 bytes): An unsigned short that MUST contain one of the values specified in the Type column in section 2.2.3.2.1.1.

Size (2 bytes): An unsigned short that MUST contain the size of the **Data** array as specified in section 2.2.3.2.1.1.

Data (variable): An array of bytes that MUST contain a security property value as specified in section 2.2.3.2.1.1.

2.2.3.2.1.1 Security Property Types

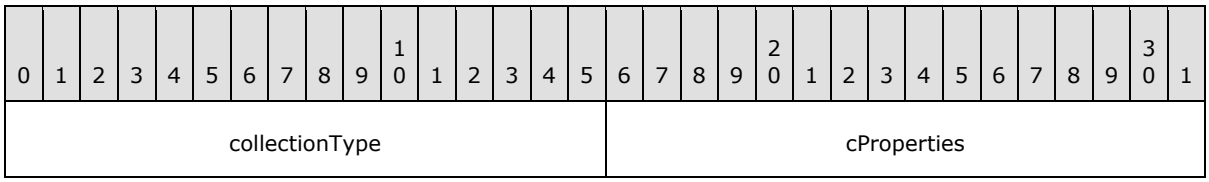
The following table lists the valid Security Property Types for the PropertyType field of the Security Property structure. See Security Property (section 2.2.3.2.1).

Type	Size field of Security Property	Data field of Security Property	Notes
0x0b01 or 0x0b06	MUST be set to the number of bytes in the Data field rounded to a multiple of 4.	MUST be an array of bytes specifying the security identifier (SID) of the caller. The array MUST be padded to a multiple of 4.	If the value is 0xb01, the **Data** field MUST contain a SID obtained by authenticating the caller using DCOM/RPC authentication mechanisms. If the value is 0x0b06, the **Data** field MUST contain a SID supplied by an application or a higher-level protocol. The **collectionType** field of the security property collection header (section 2.2.3.2.2) MUST be set to 0x0a02.
0x0b02 or 0x0b07	MUST be set to the number of bytes in the Data field rounded to a multiple of 4.	MUST be an array of Unicode characters that specifies the Windows NT account name of the caller. The array MUST be terminated with the NULL Unicode character and MUST be padded to a multiple of 4.	If the value is 0x0b02, the **Data** field MUST contain a Windows NT account name obtained by authenticating the caller using DCOM/RPC authentication mechanisms. If the value is 0x0b07, the **Data** field MUST contain a Windows NT account name supplied by an application or a higher-level protocol. The **collectionType** field of the security property collection header (section 2.2.3.2.2) MUST be set to 0x0a02.
0x0b03	MUST be set to 0x0004.	MUST be a **DWORD** that MUST contain the RPC authentication service value used in the ORPC call. For more details on RPC authentication services, see [MS-DCOM] section 2.2.1.26.2.	The **collectionType** field of the security property collection header (section 2.2.3.2.2) MUST be set to 0x0a02.
0x0b04	MUST be set to 0x0004.	MUST be a **DWORD** that MUST contain the RPC authentication level value used in the ORPC call. For more details on RPC authentication levels, see [MS-DCOM] section	The **collectionType** field of the security property collection header (section 2.2.3.2.2) MUST be set to 0x0a02.

Type	Size field of Security Property	Data field of Security Property	Notes
		2.2.1.26.1.	
0x0b05	MUST be set to 0x0004.	MUST be a **DWORD** that MUST contain the RPC impersonation level value used in the ORPC call.	The **collectionType** field of the security property collection header (section 2.2.3.2.2) MUST be set to 0x0a02.
0x0b10	MUST be set to 0x0004.	MUST be a **DWORD** that contains the minimum of the RPC authentication level values used across all the calls in the ORPC call chain. For more details on RPC authentication levels, see [MS-DCOM] section 2.2.1.26.1.	The **collectionType** field of the security property collection header (section 2.2.3.2.2) MUST be set to 0x0a01.

2.2.3.2.2 Security Property Collection Header

The Security Property Collection Header structure specifies the header of a Security Property (section 2.2.3.2.1) collection.



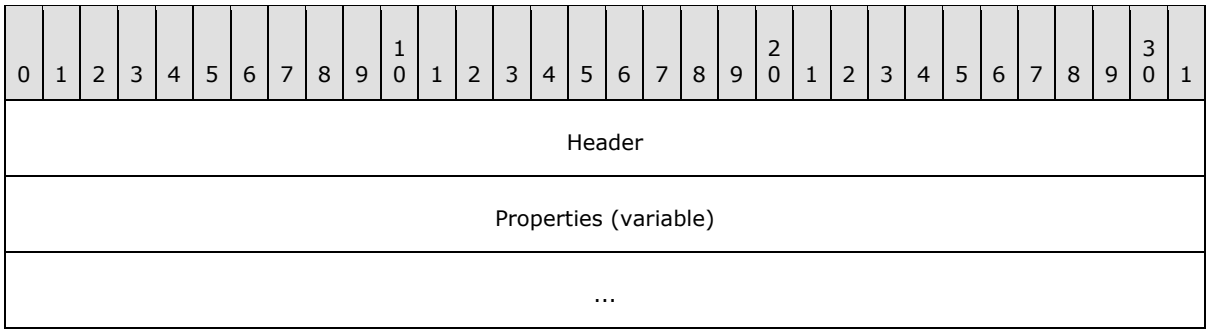
collectionType (2 bytes): An unsigned short that MUST contain one of the following values:

Value	Meaning
0x0a01	The collection MUST contain properties that are not specific to any one caller in the ORPC call chain, but that apply to the entire ORPC call chain.
0x0a02	The collection MUST contain properties that describe one caller in the ORPC call chain.

cProperties (2 bytes): An unsigned short that MUST contain the number of Security Property structures in the collection. MUST NOT be zero.

2.2.3.2.3 Security Property Collection

The Security Property Collection structure is used to specify an array of Security Property (section 2.2.3.2.1) structures. It consists of a collection header followed by the Security Property structures.



Header (4 bytes): A Security Property Collection Header (section 2.2.3.2.2).

Properties (variable): An array of Security Property structures. The number of elements in the array MUST be specified in the **cProperties** field of **Header**.

If the **collectionType** field of the **Header** has a value of 0x0a01, the **Properties** array SHOULD contain a single element with the **PropertyType** field value set to 0x0b10, specifying the minimum RPC authentication level used across the ORPC call chain.

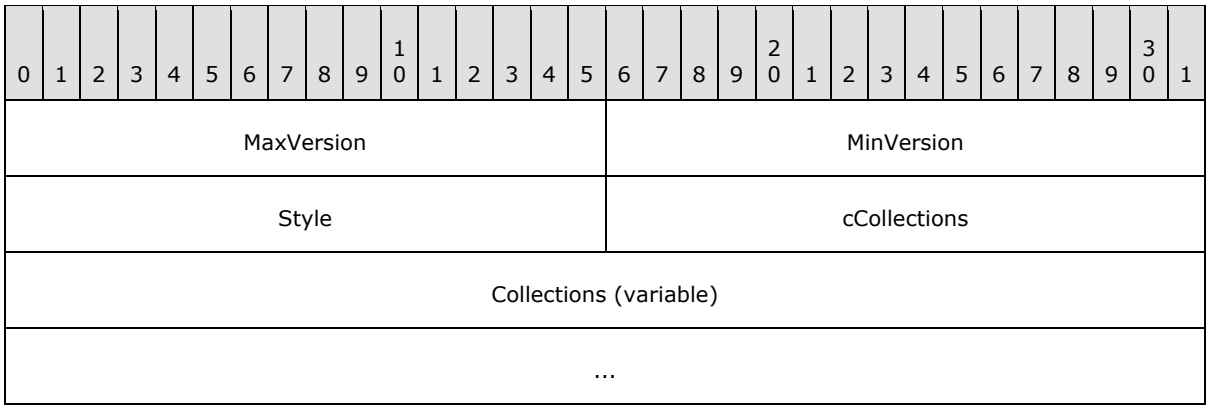
If the **collectionType** field of the **Header** has a value of 0xa02, the **Properties** array SHOULD contain at least 4 elements with the **PropertyType** values set to 0x0b01, 0x0b03, 0x0b04 and 0x0b05, specifying, respectively, the SID, the **authentication service**, the authentication level, and the **impersonation level** used in the ORPC call.

If the **collectionType** field of the **Header** has a value of 0xa02 and if the ORPC call crosses a domain boundary, the **Properties** array SHOULD contain an additional element with the **PropertyType** value set to 0xb02, specifying the Windows NT account name of the caller.

Otherwise, if the **collectionType** field of the **Header** has a value of 0xa02, if the ORPC call crosses a computer boundary and if the security identity of the client is scoped to the local computer, the **Properties** array SHOULD contain an additional element with the **PropertyType** value set to 0xb02, specifying the Windows NT account name of the caller.

2.2.3.2.4 Security ORPC Extension

The Security ORPC Extension structure is used to specify the version, style, and number of security property collections in the out-of-band data sent by the security ORPC extension.



MaxVersion (2 bytes): The major version number for this Security ORPC Extension format; this field MUST be set to 0x0001.

MinVersion (2 bytes): The minor version number for this Security ORPC Extension format; this field MUST be set to 0x0001.

Style (2 bytes): An unsigned short that MUST be set to one of the following values:

Value	Meaning
0x0000	The recipient of the ORPC call MUST append the security property collection (section 2.2.3.2.3) of the recipient when making an outgoing ORPC call.
0x0002	The recipient of the ORPC call MUST NOT append the security property collection (section 2.2.3.2.3) of the recipient when making an outgoing ORPC call.

cCollections (2 bytes): An unsigned short that MUST be set to the number of elements in the Collections array.

Collections (variable): An array of security property collections (section 2.2.3.2.3). The number of elements in the array MUST be specified in **cCollections**. The **collectionType** field in the Security Property Collection Header (section 2.2.3.2.2) of the first element in the array MUST be set to 0xa01. The **collectionType** field in the Security Property Collection Header of the remaining elements in the array MUST be set to 0xa02. The first array element with the **collectionType** field set to 0xa02 MUST specify the security property of the direct ORPC caller. Subsequent array elements MUST specify the security properties of previous callers in the ORPC call chain.

2.2.4 OBJREF_EXTENDED Context Properties

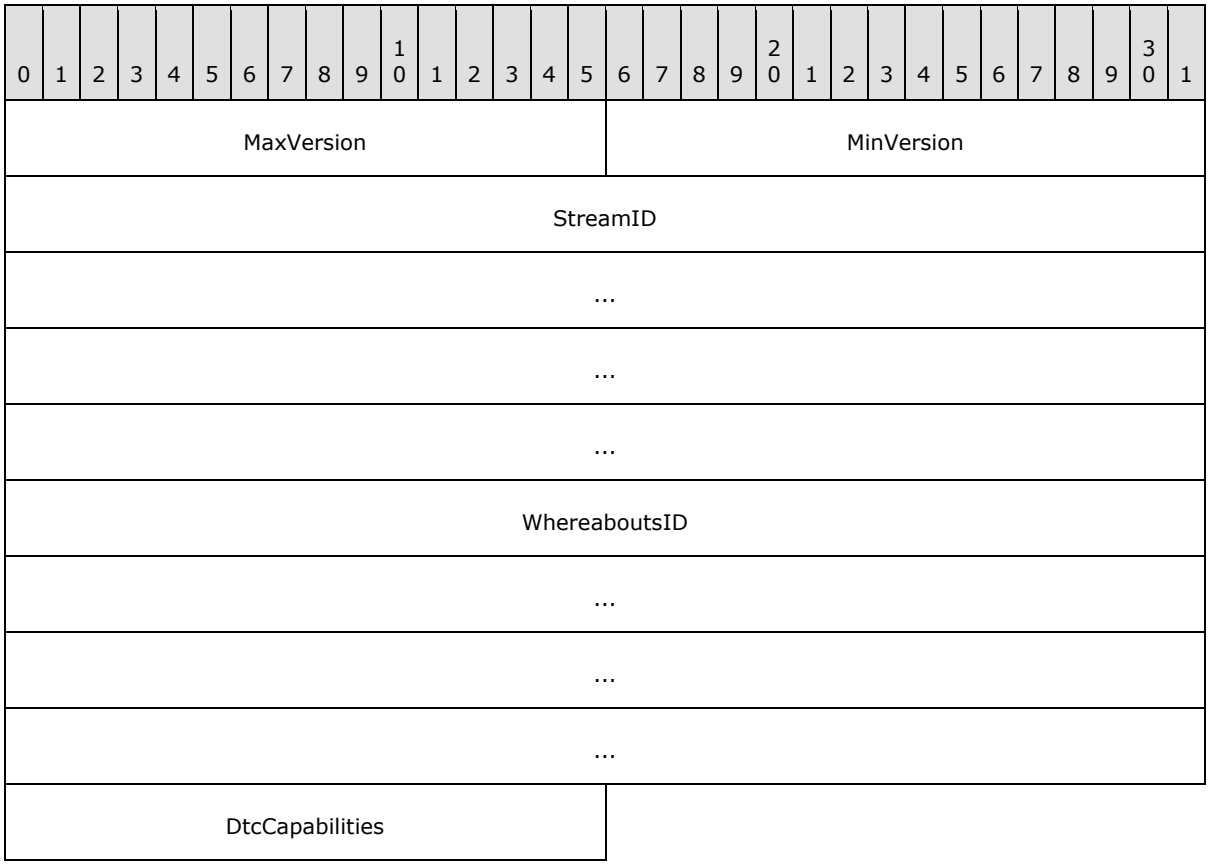
The server represents some or all server context properties as part of the marshaled OBJREF using the OBJREF_EXTENDED format ([MS-DCOM] section 2.2.1.17.7). Such properties are also known as envoy properties.

2.2.4.1 Transaction Envoy Property

The Transaction Envoy Property is used to notify the unmarshaling client that the server object is running within a transaction. The server object returns the transaction envoy context property as part of an OBJREF_EXTENDED instance.

The **policyId** field of the PROPMARSHALHEADER instance ([MS-DCOM] section 2.2.1.19.1) for the transaction envoy property MUST be set to guidTransactionProperty (see section 1.9). The **CLSID** field of the PROPMARSHALHEADER instance ([MS-DCOM] section 2.2.1.19.1) for the transaction envoy property MUST be set to CLSID_TransactionEnvoy (see section 1.9).

The marshaled data buffer for the property MUST be specified in the following format:



MaxVersion (2 bytes): The major version number for this Transaction Envoy property format; this field MUST be set to 0x0001.

MinVersion (2 bytes): The minor version number for this Transaction Envoy property format; this field MUST be set to 0x0001.

StreamID (16 bytes): A GUID that MUST contain the transaction stream id of the server.

WhereaboutsID (16 bytes): A GUID identifying the server object's SWhereabouts. For more information, see [MS-DTCO].

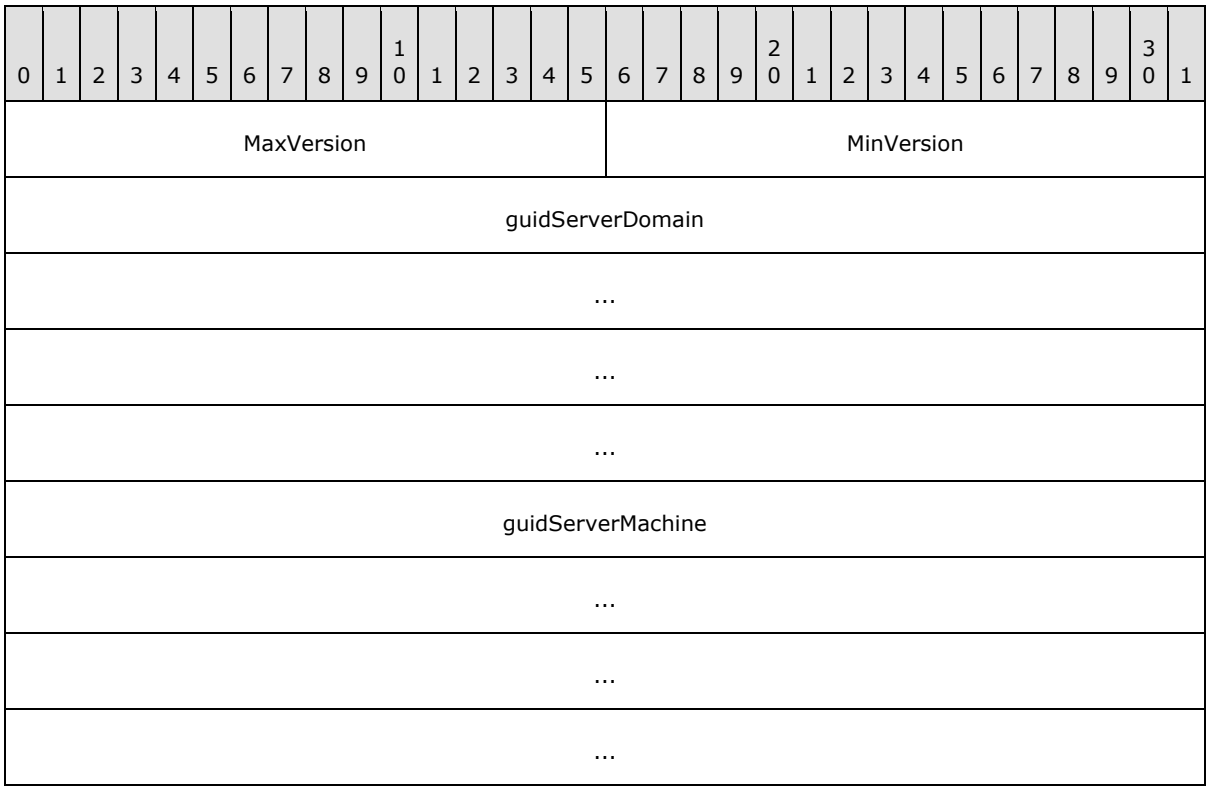
DtcCapabilities (2 bytes): An unsigned short that MUST be set to one or more of the values defined in section 2.2.6.1.

2.2.4.2 Security Envoy Property

The Security Envoy Property is used to notify the unmarshaling client that the server object is using COM+ security. The server object returns the security envoy context property as part of an OBJREF_EXTENDED instance.

The **policyId** field of the PROPMARSHALHEADER instance ([MS-DCOM] section 2.2.1.19.1) for the security envoy property MUST be set to guidSecurityProperty (see section 1.9). The **CLSID** field of the PROPMARSHALHEADER instance ([MS-DCOM] section 2.2.1.19.1) for the security envoy property MUST be set to CLSID_SecurityEnvoy (see section 1.9).

The marshaled data buffer for the property MUST be specified in the following format:



MaxVersion (2 bytes): The major version number for this security envoy property format; this field MUST be set to 0x0001.

MinVersion (2 bytes): The minor version number for this security envoy property format; this field MUST be set to 0x0001.

guidServerDomain (16 bytes): A GUID that uniquely identifies the domain of the server machine. For more information, see **[MS-NRPC]** section 2.2.1.2.1.

guidServerMachine (16 bytes): A GUID that uniquely identifies the server machine.

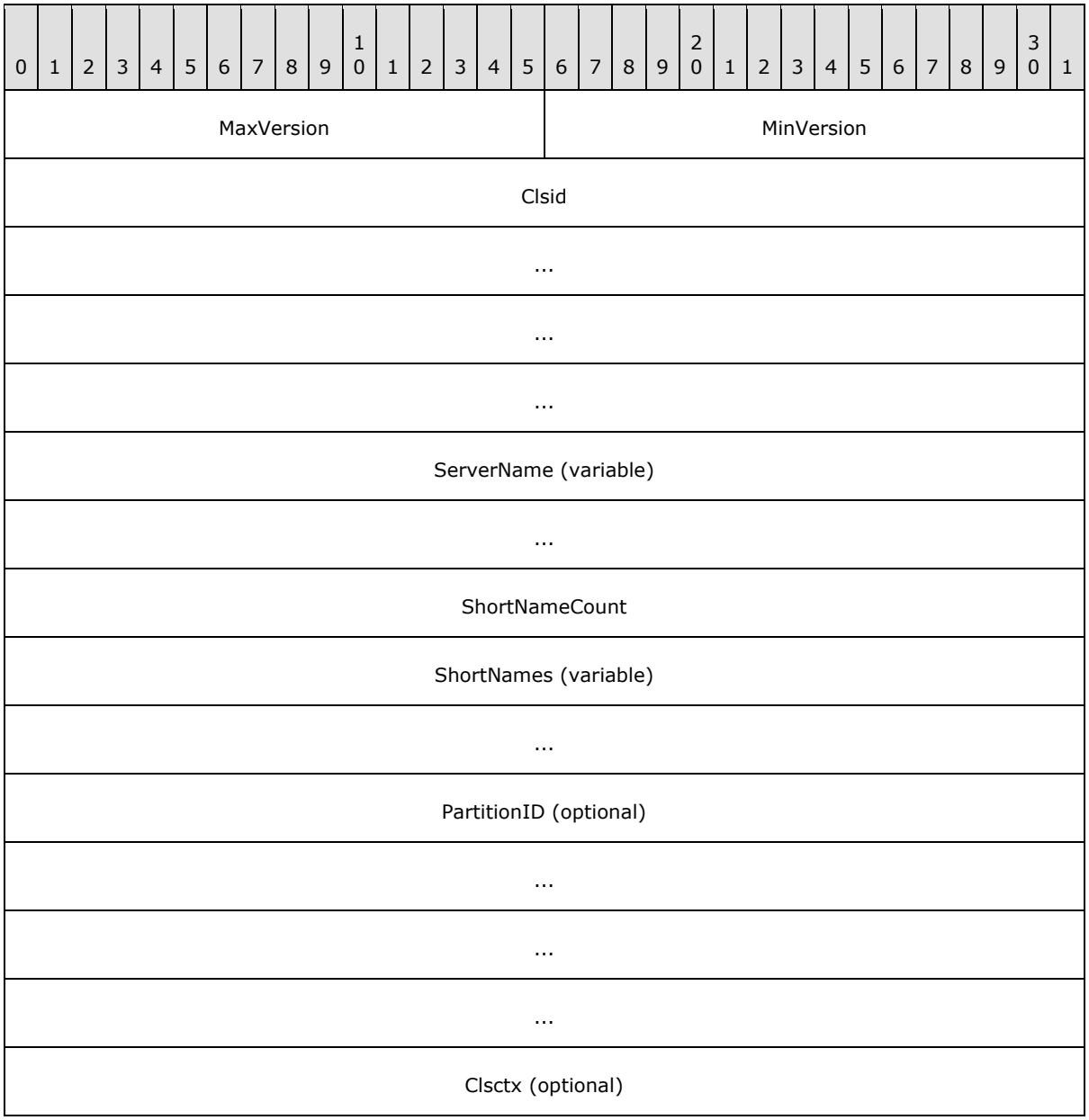
2.2.5 Class Factory Wrapper

If a client with a **COMVERSION** ([MS-DCOM] section 2.2.1.11) greater than or equal to 5.6 requests a **class factory** reference during activation ([MS-DCOM] section 3.2.2.5.2.3.2), the server

MUST return an OBJREF_CUSTOM instance containing a marshaled representation of the class factory. The unmarshaler of the OBJREF_CUSTOM instance on the client MUST convert object creation requests on the class factory reference to normal object activation requests. This process enables the client to send its client and prototype context properties during class factory based object activation requests in the same way that these properties are sent during normal object activation requests.

CLSID_CFW (see section 1.9) MUST be the unmarshaler **CLSID** for the OBJREF_CUSTOM instance.

The format of the OBJREF_CUSTOM.pObjectData buffer for this CLSID_CFW MUST be specified as follows:



BytesRemaining (optional)
LongNameCount (optional)
LongNameBytes (optional)
LongNames (variable)
...

MaxVersion (2 bytes): The major version number for this Class Factory Wrapper format; this field MUST be set to 0x0002, 0x0003, 0x0004, or 0x0005. The value indicates which fields are present, as noted in the relevant fields below.

MinVersion (2 bytes): The minor version number for this Class Factory Wrapper format; this field MUST be set to 0x0002.

Clsid (16 bytes): A CLSID is a **UUID** that MUST identify the object class of the object to be created.

ServerName (variable): A LengthPrefixedName (section 2.2.1) that contains the name of the server machine on which the object is to be created.

ShortNameCount (4 bytes): A **DWORD** that MUST specify the number of elements in the **ShortNames** array.

ShortNames (variable): An array of LengthPrefixedName (section 2.2.1) that MUST specify alternate names or addresses for the server machine on which the object is to be created. The **Length** field of each element in the array MUST be less than 16.

PartitionID (16 bytes): A GUID that MUST specify the partition id of the partition of the object class of the server object. This field MUST NOT be present if **MaxVersion** is less than 0x0003 and MUST be present otherwise.

Clsctx (4 bytes): A **DWORD** that MUST be set to the value of the **dwOrigClsCtx** field contained in the **SpecialPropertiesData** structure ([MS-DCOM] section 2.2.1.21.2.2) specified in an activation request for the class factory. This field MUST NOT be present if **MaxVersion** is less than 0x0003 and MUST be present otherwise.

BytesRemaining (4 bytes): A DWORD that MUST specify the number of bytes remaining in the buffer after the BytesRemaining field. This value MUST be equal to the sum of **LongNameBytes** plus 8. This field MUST NOT be present if **MaxVersion** is less than 0x0004 and MUST be present otherwise.

LongNameCount (4 bytes): A DWORD that MUST specify the number of elements in the **LongNames** array. This field MUST NOT be present if **MaxVersion** is less than 0x0005 and MUST be present otherwise.

LongNameBytes (4 bytes): A DWORD that MUST specify the number of bytes needed to contain all of the names contained in the **LongNames** array. This field MUST NOT be present if **MaxVersion** is less than 0x0005 and MUST be present otherwise.

LongNames (variable): An array of NULL-terminated Unicode strings that MUST specify alternate names or addresses for the server machine on which the object is to be created. This field MUST NOT be present if **MaxVersion** is less than 0x0005.

2.2.6 Constants

2.2.6.1 MS-DTC Capabilities

The constants in the following table specify the transaction propagation methods supported by an MS-DTC implementation. For more details, see [MS-DTCO].

Value	Meaning
DtcCap_CanExport (0x0001)	The specified MS-DTC implementation MUST support transaction export/import functionality via STxInfo. For more details, see [MS-DTCO].
DtcCap_CanTransmit (0x0002)	The specified MS-DTC implementation MUST support transaction transmitter/receiver functionality via Propagation_Token. For more details, see [MS-DTCO].

2.2.6.2 Transaction Isolation Levels

The constants in the following table map a subset of the isolation levels (for more details, see [MS-DTCO]) to COM+ Protocol-specific values. COM+ supports only the isolation levels listed in the following table.

Value	Corresponding ISOLATIONLEVEL value	Meaning
TxIsolationLevelReadUncommitted (0x1)	ISOLATIONLEVEL_READUNCOMMITTED	Client MUST use a read uncommitted transaction level.
TxIsolationLevelReadCommitted (0x2)	ISOLATIONLEVEL_READCOMMITTED	Client MUST use a read committed transaction level.
TxIsolationLevelRepeatableRead (0x3)	ISOLATIONLEVEL_REPEATABLEREAD	Client MUST use a repeatable read transaction level.
TxIsolationLevelSerializable (0x4)	ISOLATIONLEVEL_SERIALIZABLE	Client MUST use a serializable transaction level.

3 Protocol Details

The COM+ Protocol influences object activations in two ways:

- Clients send context properties as part of the client and/or prototype contexts.
- Servers process the context properties in the client and/or prototype contexts during the creation and configuration of server objects.

The following activation-related sections detail these operations as they pertain to the different features of the protocol.

The COM+ Protocol influences and adds special behaviors to ORPCs in several places:

- Client-side issuing of ORPCs.
- Server-side receipt of ORPCs.
- Server-side response to ORPCs.
- Client-side receipt of the server response to ORPCs.

The following ORPCs-related sections detail these operations as they pertain to the different features of the protocol.

3.1 Common Transaction Object Activation Details

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 that described in this document.

Each transactional object, whether root or non-root, MUST maintain the following data structures:

- A reference to the transaction stream supplying transactions for the object, if applicable.
- The transaction stream ID of the transaction stream, if applicable.
- The current transaction sequence number.
- A Propagation_Token and/or STxInfo instance for the currently active transaction; for more details, see [MS-DTCO].
- A Boolean value denoting whether the current transaction MUST be canceled.
- A table of transaction envoy properties associated with references to other objects in the same transaction.

In addition, a global table of **WhereaboutsID** to cached SWhereabouts SHOULD be maintained across all instances of transactional objects.

3.1.2 Timers

Not applicable to Common Transaction Object Activation (section 3.1).

3.1.3 Initialization

Not applicable to Common Transaction Object Activation (section 3.1).

3.1.4 Message Processing Events and Sequencing Rules

3.1.5 Timer Events

Not applicable to Common Transaction Object Activation (section 3.1).

3.1.6 Other Local Events

Not applicable to Common Transaction Object Activation (section 3.1).

3.2 Root Transaction Object Activation 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 that described in this document.

A root transaction object MUST maintain the data structures described in section 3.1.

3.2.2 Timers

Not applicable to Root Transaction Object Activation (section 3.2).

3.2.3 Initialization

When a root transaction object is initialized, it MUST:

- Create the data structures described in section 3.2.1.
- Detect whether the local implementation (as specified in [MS-DTCO]) supports the Propagation_Token and/or STxInfo methods of sending transactions.
- Create a transaction stream.

3.2.4 Message Processing Events and Sequencing Rules

When a root transaction object issues an object activation request, it MUST include a Transaction Context Property (section 2.2.2.1) as part of the client context.

If the client designates the server object as able to participate in a stream of transactions for future potential units of work, it MUST send a TransactionStream (section 2.2.2.1.2) structure and MUST initialize it by setting:

- The **MaxVersion** field to 0x0001 if the **IsolationLevel** field is not included; otherwise, to 0x0002.
- The **Variant** field to 0x0000.
- The **StreamID** field to the transaction stream identifier of the root transaction object's transaction stream.
- The **StreamVariant** field to 0x0001.
- The **DtcCapabilities** field to one or more of the values specified in section 2.2.2.1.2.
- The **MarshalSize** field to the size, in bytes, of the **TransactionStream** field.
- The **TransactionStream** field to contain a marshaled OBJREF for the root transaction object's transaction stream.
- The **IsolationLevel** field, if present, to one of the values specified in section 2.2.6.2.

If the client designates the server object as able to participate in only a single transaction it MUST use the TransactionBuffer (section 2.2.2.1.3) structure specified as follows:

The client MUST set:

- The **MaxVersion** field to 0x0001 if the **IsolationLevel** field is not included; otherwise, to 0x0002.
- The **Variant** field to 0x0000.
- The **StreamID** field to the transaction stream identifier of the root transaction object's transaction stream.
- The **StreamVariant** field to 0x0002.
- The **BufferSize** field to the size in bytes of the **TransactionBuffer** field.
- The **TransactionBuffer** field to contain a Propagation Token structure (as specified in [MS-DTCO] section 2.2.5.4) for the current transaction. For more details, see [MS-DTCO].
- The **IsolationLevel** field, if present, to one of the values specified in section 2.2.6.2.

3.2.5 Timer Events

Not applicable to Root Transaction Object Activation (section 3.2).

3.2.6 Other Local Events

3.2.6.1 Transaction Commit

The root transaction object MUST initiate a commit of the current transaction no later than the point of destruction of the root object.

3.2.6.2 Transaction Abort

The root transaction object MUST initiate a cancellation of the current transaction, if the TransactionPropRetFlag_Abort flag has been returned by an ORPC call to a non-root transactional object, or if so requested by a higher-level protocol.

3.3 Non-Root Transaction Object Activation 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 that described in this document.

A non-root transaction object MUST maintain the data structures described in section 3.1. In addition, a non-root transaction object MUST maintain the following data structures:

- The isolation level of the transaction stream, if supplied by the client object during activation. If the client object does not specify an isolation level, it MUST be assumed to be TxIsolationLevelSerializable.

3.3.2 Timers

Not applicable to Non-root Transaction Object Activation (section 3.3).

3.3.3 Initialization

When a non-root transaction object is initialized, it MUST:

- Create the data structures described in 3.3.1.
- Detect whether the local [MS-DTCO] implementation supports Propagation_Token and/or STxInfo methods of sending transactions.
- Register as a transaction voter or resource manager (see [MS-DTCO]) with its local [MS-DTCO] implementation.

3.3.4 Message Processing Events and Sequencing Rules

When a non-root transaction object issues an object activation request, it MUST include a Transaction Context Property (section 2.2.2.1) as part of the client context. If the client object supplies a transaction stream reference during activation, the activation MUST use the TransactionStream (section 2.2.2.1.2) format; otherwise, it MUST use the TransactionBuffer (section 2.2.2.1.3) format.

3.3.5 Timer Events

Not applicable to Non-root Transaction Object Activation (section 3.3).

3.3.6 Other Local Events

3.3.6.1 Transaction Outcome Participation

Non-root transaction objects do not control transaction lifetimes, but MUST be prepared to vote on the outcome of each transaction in which they participate, when so requested by their local [MS-DTCO] implementation. A non-root transaction object MUST vote to cancel a transaction if the TransactionPropRetFlag_Abort flag has been returned by an ORPC call, or if so requested by a higher-level protocol.

3.4 Client Activity Activation 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 that described in this document.

A client object running within an activity MUST maintain the following data structures:

- An activity identifier GUID.
- An activity time-out value.

3.4.2 Timers

Not applicable to client activity activation.

3.4.3 Initialization

Not applicable to client activity activation.

3.4.4 Message Processing Events and Sequencing Rules

If the client object is running in a context with an activity, the client MUST create an activity context property (see section 2.2.2.2) as part of the client context in the activation request. It MUST set the ActivityID field to the activity identifier of the client object's activity. It MUST set the TimeOut field to the activity time-out of the client object's activity.

3.4.5 Timer Events

Not applicable to client activity activation.

3.4.6 Other Local Events

Not applicable to client activity activation.

3.5 Client Partition Activation Details

3.5.1 Abstract Data Model

Not applicable to Client Partition Activation (section 3.5).

3.5.2 Timers

Not applicable to Client Partition Activation (section 3.5).

3.5.3 Initialization

Not applicable to Client Partition Activation (section 3.5).

3.5.4 Message Processing Events and Sequencing Rules

When a client object issues an object activation, the client MUST specify a partition id in the **guidPartition** field of the **SpecialPropertiesData** structure ([MS-DCOM] section 2.2.1.21.2.2). The specified partition id MUST be ONE of the following:

- GUID_NULL, to request that the server select a partition for the client.
- The partition ID associated with the client object context.
- The ID of the partition that the client requires the server object to be configured in.

3.5.5 Timer Events

Not applicable to Client Partition Activation (section 3.5).

3.5.6 Other Local Events

Not applicable to Client Partition Activation (section 3.5).

3.6 Client User Property Activation 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 as long as their external behavior is consistent with that described in this document.

A client user context property MUST maintain the following data structures:

- A table of name/value pair mappings, with the types of names and values as specified in section 2.2.2.3.

3.6.2 Timers

Not applicable to Client User Property Activation (section 3.6).

3.6.3 Initialization

Not applicable to Client User Property Activation (section 3.6).

3.6.4 Message Processing Events and Sequencing Rules

If an application or higher-level protocol supplies user-defined context properties (see section 2.2.2.3) during activation, the client MUST copy and propagate them as part of both the client context and the prototype context in the activation request.

3.6.5 Timer Events

Not applicable to Client User Property Activation (section 3.6).

3.6.6 Other Local Events

Not applicable to Client User Property Activation (section 3.6).

3.7 Client Class Factory Wrapper Activation 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.

The client MUST maintain the following data structures

- A class factory wrapper (section 2.2.5).

3.7.2 Timers

Not applicable to Client Class Factory Wrapper Activation (section 3.7).

3.7.3 Initialization

On initialization, the client MUST:

- Unmarshal the OBJREF_CUSTOM instance contained in the response to the activation request for a class factory object.
- Initialize the class factory wrapper (section 2.2.5) structure with the corresponding fields from the class factory wrapper structure contained in the **pObjectData** field of the OBJREF_CUSTOM instance.

3.7.4 Message Processing Events and Sequencing Rules

When the application makes object creation requests on the class factory object reference, the client MUST:

- Make an object activation request ([MS-DCOM] sections 3.1.4.1.1 and 3.2.2.5.2) by:
  - Specifying the value of the **ServerName** field from the class factory wrapper (section 2.2.5) structure as the remote server name for the activation request.
  - Setting the value of the **classID** field in the **InstantiationInfoData** structure ([MS-DCOM] section 2.2.1.21.2.1) to the value of the **Clsid** field from the class factory wrapper structure.
  - Setting the value of the **guidPartition** field in the **SpecialPropertiesData** structure ([MS-DCOM] section 2.2.1.21.2.2) to the value of the **PartitionID** field from the class factory wrapper structure.
  - Setting the value of the **dwOrigClsCtx** field in the **SpecialPropertiesData** structure ([MS-DCOM] section 2.2.1.21.2.2) to the value of the **Clsctx** field from the class factory wrapper structure.
  - Sending client and prototype context properties in the **ActivationContextInfoData** structure ([MS-DCOM] section 2.2.1.21.2.5) as specified in section 1.3.1.1.

- If the activation request succeeds, return success, or continue processing as follows if not.
- For each element in the **ShortNames** array in the class factory wrapper structure:
  - Issue an activation request, as specified above, except that the client MUST specify the alternate server name or address contained in the **ShortNames** array as the remote server name for the activation request.
  - If the activation request succeeds, return success; otherwise, continue processing.
- For each element in the **LongNames** array in the class factory wrapper structure:
  - Issue an activation request, as specified above, except that the client MUST specify the alternate server name or address contained in the **LongNames** array as the remote server name for the activation request.
  - If the activation request succeeds, return success; otherwise, continue processing.
- Return the error code from the last activation request to the application or higher layer protocol.

3.7.5 Timer Events

Not applicable to Client Class Factory Wrapper Activation (section 3.7).

3.7.6 Other Local Events

Not applicable to Client Class Factory Wrapper Activation (section 3.7).

3.8 Server Transaction Activation 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.

Server transaction objects MUST maintain the data structures described in either section 3.2.1 or section 3.3, as applicable.

3.8.2 Timers

Not applicable to Server Transaction Activation (section 3.8).

3.8.3 Initialization

Not applicable to Server Transaction Activation (section 3.8).

3.8.4 Message Processing Events and Sequencing Rules

When processing an activation, the server MUST decide in an implementation-specific way whether the server object is to be a root transaction object, a non-root transaction object, or a non-transactional object.

If the server object is to be a root transaction object, the server MUST:

- Create a transaction stream.
- Initiate communication with the local [MS-DTCO] implementation.
- Create the object using an implementation-specific mechanism.
- Marshal the object as described in section 3.18.

If the server object is to be a non-root transaction object:

- If the client supplied a TransactionStream as part of the activation, the server MUST:
  - Save the transaction stream reference.
  - Initiate communication with the local [MS-DTCO] implementation.
  - Register as a voter or resource manager with [MS-DTCO].
  - Create the object using an implementation-specific mechanism.
  - Marshal the object as described in section 3.18, and return it to the client.
- If the client supplied a TransactionBuffer as part of the activation, the server MUST:
  - Save the supplied Propagation Token structure (as specified in [MS-DTCO] section 2.2.5.4).
  - Initiate communication with the local [MS-DTCO] implementation.
  - Register as a voter or resource manager with [MS-DTCO].
  - Create the object using an implementation-specific mechanism.
  - Marshal the object as described in section 3.18 and return it to the client.
- Otherwise, the server object is not a child of a root transaction.

If the server object is to be a non-transactional object, the server MUST:

- Create the object using an implementation-specific mechanism.
- Marshal the object as an OBJREF ([MS-DCOM] section 2.2.1.17) of type OBJREF_STANDARD ([MS-DCOM] section 2.2.1.17.4) and return it to the client.

3.8.5 Timer Events

Not applicable to Server Transaction Activation (section 3.8).

3.8.6 Other Local Events

Not applicable to Server Transaction Activation (section 3.8).

3.9 Server Activity Activation Details

3.9.1 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 explain 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.

The server MUST maintain the following data structures per object:

- An activity identifier GUID.
- An activity time-out value.

3.9.2 Timers

Not applicable to Server Activity Activation (section 3.9).

3.9.3 Initialization

Not applicable to Server Activity Activation (section 3.9).

3.9.4 Message Processing Events and Sequencing Rules

When processing an activation, the server MUST decide, in an implementation-specific way, if the object is to share the client's activity, run in a new activity, or not use an activity at all.

If the object is to share the client's activity, the server MUST:

- Set the activity ID and activity time-out to the values from the corresponding fields in the client's activity context property (see section 2.2.2.2).

If the object is to run in a new activity, the server MUST:

- Create a new activity ID GUID.
- Set the activity time-out to an implementation-specific value.<2>

If the server object is to run without an activity, the server MUST NOT associate activity data with the object.

3.9.5 Timer Events

Not applicable to Server Activity Activation (section 3.9).

3.9.6 Other Local Events

Not applicable to Server Activity Activation (section 3.9).

3.10 Server Partition Activation Details

3.10.1 Abstract Data Model

Not applicable to Server Partition Activation (section 3.10).

3.10.2 Timers

Not applicable to Server Partition Activation (section 3.10).

3.10.3 Initialization

Not applicable to Server Partition Activation (section 3.10).

3.10.4 Message Processing Events and Sequencing Rules

When processing an activation request, the server MUST do the following:

- If the partition ID specified by the client in the guidPartition field of the **SpecialPropertiesData** structure ([MS-DCOM] section 2.2.1.21.2.2) is GUID_NULL, the server MUST select a partition for the server object in an implementation-specific manner. If a partition cannot be determined in an implementation-specific manner, the server MUST select the global partition.
- If the partition ID specified by the client in the guidPartition field of the **SpecialPropertiesData** structure ([MS-DCOM] section 2.2.1.21.2.2) is not GUID_NULL, the server MUST select the partition specified by the partition ID. If the partition does not exist, the server MUST return an error to the activation request.

3.10.5 Timer Events

Not applicable to Server Partition Activation (section 3.10).

3.10.6 Other Local Events

Not applicable to Server Partition Activation (section 3.10).

3.11 Server User Property Activation Details

3.11.1 Abstract Data Model

Not applicable to Server User Property Activation (section 3.11).

3.11.2 Timers

Not applicable to Server User Property Activation (section 3.11).

3.11.3 Initialization

Not applicable to Server User Property Activation (section 3.11).

3.11.4 Message Processing Events and Sequencing Rules

When processing an activation, if user-defined context properties (section 2.2.2.3) are present in the client and prototype contexts, the server MUST copy and supply these properties to applications or higher-level protocols that consume the properties.

3.11.5 Timer Events

Not applicable to Server User Property Activation (section 3.11).

3.11.6 Other Local Events

Not applicable to Server User Property Activation (section 3.11).

3.12 Server Class Factory Wrapper Activation Details

3.12.1 Abstract Data Model

Not applicable to Server Class Factory Wrapper Activation (section 3.12).

3.12.2 Timers

Not applicable to Server Class Factory Wrapper Activation (section 3.12).

3.12.3 Initialization

Not applicable to Server Class Factory Wrapper Activation (section 3.12).

3.12.4 Message Processing Events and Sequencing Rules

If the activation request is for a class factory object ([MS-DCOM] section 3.2.2.5.2.3.2), and if the **COMVERSION** ([MS-DCOM] section 2.2.1.11) of the client is greater than or equal to 5.6, the server MUST:

- Create an OBJREF_CUSTOM instance ([MS-DCOM] section 2.2.1.17.6) for the marshaled object reference of the class factory object.
- Create and initialize the **pObjectData** field of the OBJREF_CUSTOM instance, and MUST set:
  - The **MaxVersion** field to 0x0005.
  - The **MinVersion** field to 0x0002.
  - The **Clsid** field to the GUID of the object class.
  - The **ServerName** field to a LengthPrefixedName (section 2.2.1) containing the computer name of the server machine.<3>
  - The **ShortNameCount** to the number of elements in the **ShortNames** array.
  - The **ShortNames** field to an array of LengthPrefixedName structures. The array MUST contain **ShortNameCount** elements. The **Length** field of each LengthPrefixedName structure MUST be less than 16. The **Name** field of each LengthPrefixedName structure MUST contain an alternate computer name or a network address of the server machine.<4>
  - Further, the server MUST set:
  - The **PartitionID** guid to the partition ID of the object class.
  - The **Clsctx** field to the value of the **dwOrigClsctx** field contained in the **SpecialPropertiesData** structure ([MS-DCOM] section 2.2.1.21.2.2) specified in the activation request for the class factory object.
  - The **BytesRemaining** field to the number of bytes in the **LongNames** array plus 8.
  - The **LongNameCount** field to the number of elements in the **LongNames** array.
  - The **LongNames** field to an array of Unicode strings. Each element in the array MUST contain an alternate computer name or a network address of the server machine.<5>

3.12.5 Timer Events

Not applicable to Server Class Factory Wrapper Activation (section 3.12).

3.12.6 Other Local Events

Not applicable to Server Class Factory Wrapper Activation (section 3.12).

3.13 Client Transaction ORPC Extension Details

3.13.1 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 explain 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.

The client MUST maintain the data structures described in either section 3.2.1 or section 3.3.1, as applicable.

3.13.2 Timers

Not applicable to Client Transaction ORPC Extensions (section 3.13).

3.13.3 Initialization

Not applicable to Client Transaction ORPC Extensions (section 3.13).

3.13.4 Message Processing Events and Sequencing Rules

When a transactional object issues an ORPC, it MUST perform the following sequence of operations:

If any of the following is TRUE, then the client MUST NOT send a Transaction ORPC Context Extension (section 2.2.3.1) on the call:

- A transaction envoy property (section 2.2.4.1) was not received from the server during unmarshaling.
- A transaction envoy property was received from the server during unmarshaling, but its transaction stream ID did not match that of the client's.
- A transaction envoy property was received from the server during unmarshaling, but the client and server do not support [MS-DTCO] implementations that can share transactions (see section 1.3.2.5).

Otherwise, the client transaction envoy MUST:

- Construct a TransactionPropCallHeader (section 2.2.3.1.1.1) and MUST set:
  - The **m_ulSeq** field to the TSN of the current transactions in use by the client object.
  - The **m_usFlags** field to zero, if either of the following conditions is TRUE:
    - The client and server are both using the Propagation_Token mechanism (as specified in section 1.3.2.5).

- The server has previously returned its SWhereabouts instance (as specified in [MS-DTCO]) in response to a previous ORPC call.

- Otherwise, the client transaction envoy MUST set the **m_usFlags** field to TransactionPropCallFlag_NeedWhereabouts to request that the server return its SWhereabouts instance (as specified in [MS-DTCO]) in the ORPC response.

- If the current TSN of the client object transaction matches the last known good TSN of the server object, and the server object has returned TransactionPropRetFlag_DontSend in response to a previous ORPC call, the client transaction envoy MUST set the **m_usVariant** field to TransactionPropCall_None.

- If the current TSN of the client object transaction does not match the last known good TSN of the server object, and client and server are using the [MS-DTCO] STxInfo transaction propagation method, the client transaction envoy (section 3.20) MUST set the **m_usVariant** field to TransactionPropCall_ExportCookie, and MUST follow the TransactionPropCallHeader structure with a TransactionPropCallExportCookie (section 2.2.3.1.1.2) containing the [MS-DTCO] STxInfo for the current transaction.

- If the current TSN of the client object transaction does not match the last known good TSN of the server object, and the client and server are using the [MS-DTCO] Propagation Token (section 2.2.5.4) transaction propagation method, the client transaction envoy MUST set the **m_usVariant** field to TransactionPropCall_TransmitterBuffer, and MUST follow the TransactionPropCallHeader structure with a TransactionPropCallTransmitterBuffer (section 2.2.3.1.1.3) containing the [MS-DTCO] Propagation Token (section 2.2.5.4) for the current transaction.

Upon return of the ORPC call, the client transaction envoy MUST process the returned TransactionPropRetHeader (section 2.2.3.1.2.1) from the server object response, as follows:

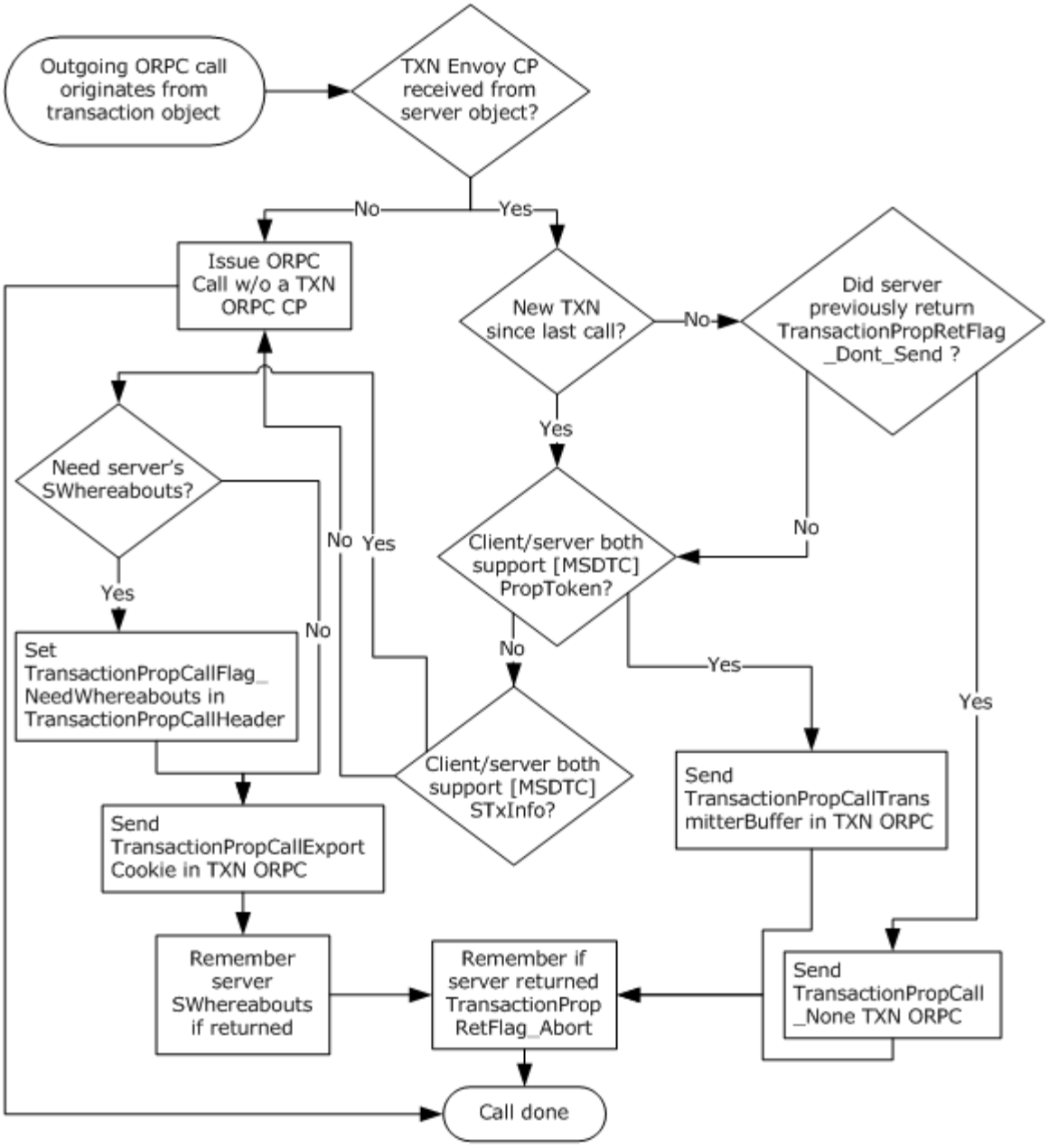
- If the **m_usFlags** field of the TransactionPropRetHeader contains the TransactionPropRetFlag_Abort flag, the client transaction envoy MUST instruct the client object either to cancel the transaction (if it is the root transaction object), or eventually to vote to abort the transaction (if it is a non-root transaction object).

- If the **m_usFlags** field of the TransactionPropRetHeader contains the TransactionPropRetFlag_DontSend flag, the client transaction envoy MUST update the last known good TSN for the server object to the same value as the TSN sent in the **m_ulSeq** of the TransactionPropCallHeader on the call.

- If the **m_usVariant** field of the TransactionPropRetHeader contains the TransactionPropRet_Whereabouts flag, then the client transaction envoy MUST add the [MS-DTCO] SWhereabouts in the following TransactionPropRetWhereabouts structure (see section 2.2.3.1.2.2) to its global table of WhereaboutsIDs-to-[MS-DTCO]-Whereabouts mappings.

3.13.4.1 Diagram

This diagram shows the logical processing flow when issuing an ORPC from a client transactional object to a server object, which might or might not be running in the same transaction as the client object.



Flow chart of outgoing ORPC from a transaction object

Figure 6: Flow chart of outgoing ORPC from a transactional object

3.13.5 Timer Events

Not applicable to Client Transaction ORPC Extensions (section 3.13).

3.13.6 Other Local Events

Not applicable to Client Transaction ORPC Extensions (section 3.13).

3.14 Client Security ORPC Extension Details

3.14.1 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 explain 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.

If the client receives an ORPC, it MUST act as a server and maintain the data structures specified in section 3.16.1. In addition, it MUST also maintain the data structures specified in section 3.21.1.

3.14.2 Timers

Not applicable to Client Security ORPC Extensions (section 3.14).

3.14.3 Initialization

Not applicable to Client Security ORPC Extensions (section 3.14).

3.14.4 Message Processing Events and Sequencing Rules

When the client makes an ORPC request (see [MS-DCOM] section 3.1.4.2), the extension MUST add a Security ORPC Extension (section 2.2.3.2.4) structure to the ORPC message. The extension MUST create the structure and MUST:

- Set the **Style** field to 0x0000 unless an application or a higher-level protocol requires the extension to set this field to 0x0002.
- If the **Style** field is set to 0x0000, the extension MUST create the rest of the structure as follows:
  - If the client is processing an incoming ORPC, the extension MUST lookup the Security ORPC Extension structure corresponding to the incoming ORPC in the Security ORPC Extension table. It MUST increment the value of the **cCollections** field in the structure by 1 and set this value in the **cCollections** field.
  - Otherwise, the extension MUST set the **cCollection** field to 0x0002.
  - Next, the extension MUST create the first collection and MUST:
    - Set the **collectionType** field in the header to 0x0a01.
    - Set the **cProperties** field in the header to 0x0001.
    - Create the Security Property (section 2.2.3.2.1) for the collection and MUST:
      - Set the **Type** field 0x0b10.
      - Set the **Size** field to 0x0004.
      - If the client is processing an incoming ORPC, the extension MUST:
        - Look up the Security ORPC Extension structure corresponding to the incoming ORPC in the Security ORPC Extension table.
        - Look up the Security Property value for type 0x0b10 in the Security ORPC Extension structure.

        - Compare this value to the authentication level of the outgoing ORPC request.
        - Set the **Data** field to the minimum of the two values.
      - Otherwise, the extension MUST set the **Data** field to the authentication level of the current ORPC call.
  - Next, if the client is processing an incoming ORPC, the extension MUST:
    - Look up the Security ORPC Extension structure corresponding to the incoming ORPC in the Security ORPC Extension table.
    - For each Security Property Collection in the Security ORPC Extension structure with a **collectionType** field in the header set to 0xa02, the extension MUST:
      - Copy the Security Property Collection to the Security ORPC Extension structure of the outgoing ORPC.
  - Next, the extension MUST add the last collection and MUST:
    - Set the **collectionType** field in the header to 0x0a02.
    - Set the **cProperties** field in the header to 0x0003.
    - Create a Security Property for the collection and MUST set:
      - The **Type** field to 0x0b03.
      - The **Size** field to 0x0004.
      - The **Data** field to the authentication service of the current ORPC call.
    - The extension MUST next create a Security Property for the collection and MUST set:
      - The **Type** field to 0x0b04.
      - The **Size** field to 0x0004.
      - The **Data** field to the authentication level of the current ORPC call.
    - The extension MUST next create a Security Property for the collection and MUST set:
      - The **Type** field to 0x0b05.
      - The **Size** field to 0x0004.
      - The **Data** field to the impersonation level of the current ORPC call.
    - If the extension can obtain the Security Identifier (SID) of the caller via authentication or from an application or higher-level protocol, it MUST:
      - Increment the **cProperties** field by 0x0001.
      - Create a Security Property for the collection and MUST set:
        - The **Type** field to 0xb01, unless an application or a higher-level protocol requires that the extension set the **Type** field to 0xb06.
        - The **Size** field to the size in bytes of the SID.

        - The **Data** field to the SID.
    - Next the extension MUST compare the **domainGUID** of the client computer to that of the object reference. If they are different, it MUST:
        - Determine, in an implementation-specific manner or from an application or a higher-level protocol, the Windows NT account name of the caller.
        - Increment the **cProperties** field by 0x0001.
        - Create a Security Property for the collection and MUST set:
            - The **Type** field to 0xb02, unless an application or a higher-level protocol requires that the extension set the **Type** field to 0xb07.
            - The **Size** field to the size, in bytes, of the **Data** field.
            - The **Data** field to the Windows NT account name of the caller.
    - Otherwise, the extension MUST compare the **machineGUID** of the client computer to that of the object reference. If they are different, it MUST:
        - Determine, in an implementation-specific manner, if the security identity of the caller is scoped within the client computer (for example, if it is a local machine account).
        - If the caller is a local machine account, the extension MUST:
            - Determine, in an implementation-specific manner or from an application or a higher-level protocol, the Windows NT account name of the caller.
            - Increment the **cProperties** field by 0x0001.
            - Create a Security Property for the collection and MUST set:
                - The **Type** field to 0xb02, unless an application or a higher level protocol requires that the extension set the **Type** field to 0xb07.
                - The **Size** field to the size, in bytes, of the **Data** field.
                - The **Data** field to the Windows NT account name of the caller.
- If the **Style** field is set to 0x0002, the extension MUST create the rest of the structure as follows:
    - If the client is currently processing an incoming ORPC, the extension MUST:
        - Look up the Security ORPC Extension structure corresponding to the incoming ORPC in the Security ORPC Extension table.
        - Set the rest of the Security ORPC Extension structure in the outgoing ORPC to the corresponding data from the Security ORPC Extension structure of the incoming ORPC.
    - Otherwise, the extension MUST set the **cCollections** field to 0x0000.

3.14.5 Timer Events

Not applicable to Client Security ORPC Extensions (section 3.14).

3.14.6 Other Local Events

Not applicable to Client Security ORPC Extensions (section 3.14).

3.15 Server Transaction ORPC Extension Details

3.15.1 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 explain 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.

The server MUST maintain the data structures described in either section 3.2.1 or section 3.3.1, as applicable.

3.15.2 Timers

Not applicable to Server Transaction ORPC Extensions (section 3.15).

3.15.3 Initialization

Not applicable to Server Transaction ORPC Extensions (section 3.15).

3.15.4 Message Processing Events and Sequencing Rules

A transaction server object that receives an ORPC MUST perform the following sequence of operations:

If the incoming ORPC does not contain a Transaction ORPC Context Extension (section 2.2.3.1), the server object MUST do the following:

- If the server and client both support Propagation_Token (for more details, see [MS-DTCO]), then the server object MUST call the *GetSeqAndTxViaTransmitter* (see section 3.22.4.2) method of its transaction stream object, specifying arguments as follows:
  - MUST set *ulKnownSeq* to the currently known TSN.

  Upon return from the *GetSeqAndTxViaTransmitter* call, the server object MUST:
  - Update its current known TSN with the one returned in *pulCurrentSeq*.
  - Update its current transaction with the one returned in *prgbTransmitterBuffer*.
- If the server and client do not both support Propagation_Token, but both do support STxInfo (for more details, see [MS-DTCO]), the server object MUST call the *GetSeqAndTxViaExport* (see section 3.22.4.1) method of its transaction stream object, specifying arguments as follows:
  - MUST set *ulKnownSeq* to the currently known TSN.
  - MUST set *ulcbWhereabouts* to the size in bytes of the SWhereabouts contained in the *rgbWhereabouts* field (as specified in [MS-DTCO]).
  - MUST set *rgbWhereabouts* to contain a SWhereabouts for the server (as specified in [MS-DTCO]).

Upon return of the *GetSeqAndTxViaExport* call, the server object MUST:

- Update its current known TSN with the one returned in *pulCurrentSeq*.
- Update its current transaction with the one returned in *prgbExportCookie*.

- Upon successful confirmation or updating of the current transaction, the server object MUST:
  - Register as a voter or resource manager if a new transaction was returned from the transaction stream (as specified in [MS-DTCO]).
- Execute the ORPC call.
- Return.

If the incoming ORPC does contain a Transaction ORPC Context Extension, the server object MUST do the following:

- If the incoming TSN (in the **m_ulSeq** field of the TransactionPropCallHeader (section 2.2.3.1.1.1)) is different from the current known TSN, the server object MUST do the following:
  - If the previous transaction has not yet been completed, the server object MUST vote to abort it when requested by [MS-DTCO].
  - If the server and client both support Propagation_Token, then the server object MUST call the *GetTxViaTransmitter* (see section 3.22.4.4) method of its transaction stream object, specifying arguments as follows:
    - MUST set *ulRequestSeq* to the TSN contained in the m_ulSeq field of the TransactionPropCallHeader.

    Upon successful return of the *GetTxViaTransmitter* call, the server MUST:

    - Update its current known TSN to the TSN contained in the **m_ulSeq** field of the TransactionPropCallHeader.
    - Update its current transaction with the one returned in *prgbTransmitterBuffer*.
  - If the server and client do not both support Propagation_Token, but both do support STxInfo, then the server object MUST call the *GetTxViaExport* (see section 3.22.4.3) method of its transaction stream object, specifying arguments as follows:
    - MUST set *ulRequestSeq* to the TSN contained in the **m_ulSeq** field of the TransactionPropCallHeader.
    - MUST set *ulcbWhereabouts* to the size in bytes of the SWhereabouts contained in the *rgbWhereabouts* field (as specified in [MS-DTCO]).
    - MUST set *rgbWhereabouts* to contain a [MS-DTCO] SWhereabouts for the server.

    Upon successful return of the *GetTxViaExport* call, the server MUST:

    - Update its current known TSN to the TSN contained in the **m_ulSeq** field of the TransactionPropCallHeader.
    - Update its current transaction with the one returned in *prgbExportCookie*.
- Upon a successful update of the current transaction the server object MUST:

- Register as a voter or resource manager with for the new transaction (as specified in [MS-DTCO]).

Then:

- Execute the ORPC call.

After the ORPC call is executed the server object MUST:

- If the client requested the server's SWhereabouts (as specified in [MS-DTCO]) by setting the TransactionPropCallFlag_NeedWhereabouts flag in the **m_usFlags** field of the TransactionPropCallHeader, the server MUST return a TransactionPropRetWhereabouts (section 2.2.3.1.2.2) structure to the client, and MUST set:
  - The **m_usFlags** field to TransactionPropRetFlag_DontSend if the current transaction was successfully updated, or to TransactionPropRetFlag_Abort if a call to a child object returned TransactionPropRetFlag_Abort, or if a higher-level protocol has directed that the current transaction be aborted.
  - The **m_usVariant** field to TransactionPropRet_Whereabouts.
  - The **cbWhereabouts** field to the size in bytes of the SWhereabouts (as specified in [MS-DTCO]) contained in **Whereabouts**.
  - The **Whereabouts** field to contain a SWhereabouts for the server's [MS-DTCO].
- If the **m_usFlags** field of the TransactionPropCallHeader is 0x0000, the server MUST return a TransactionPropRetHeader (section 2.2.3.1.2.1) structure to the client, and MUST set:
  - The **m_usFlags** field to TransactionPropRetFlag_DontSend if the current transaction was successfully updated, or to TransactionPropRetFlag_Abort if a call to a child object returned TransactionPropRetFlag_Abort, or if a higher-level protocol has directed that the current transaction be canceled.
  - The **m_usVariant** field to TransactionPropRet_None.

3.15.4.1 Diagram

The following diagram shows the logical processing flow when a transactional server object processes an ORPC. The ORPC originates either from another an object running in the same transaction as the server object, or from an object or a client not associated with the server object's transaction.

The following figure shows a flowchart of an incoming ORPC to a non-root transaction object. TXN CP is a transaction context property.

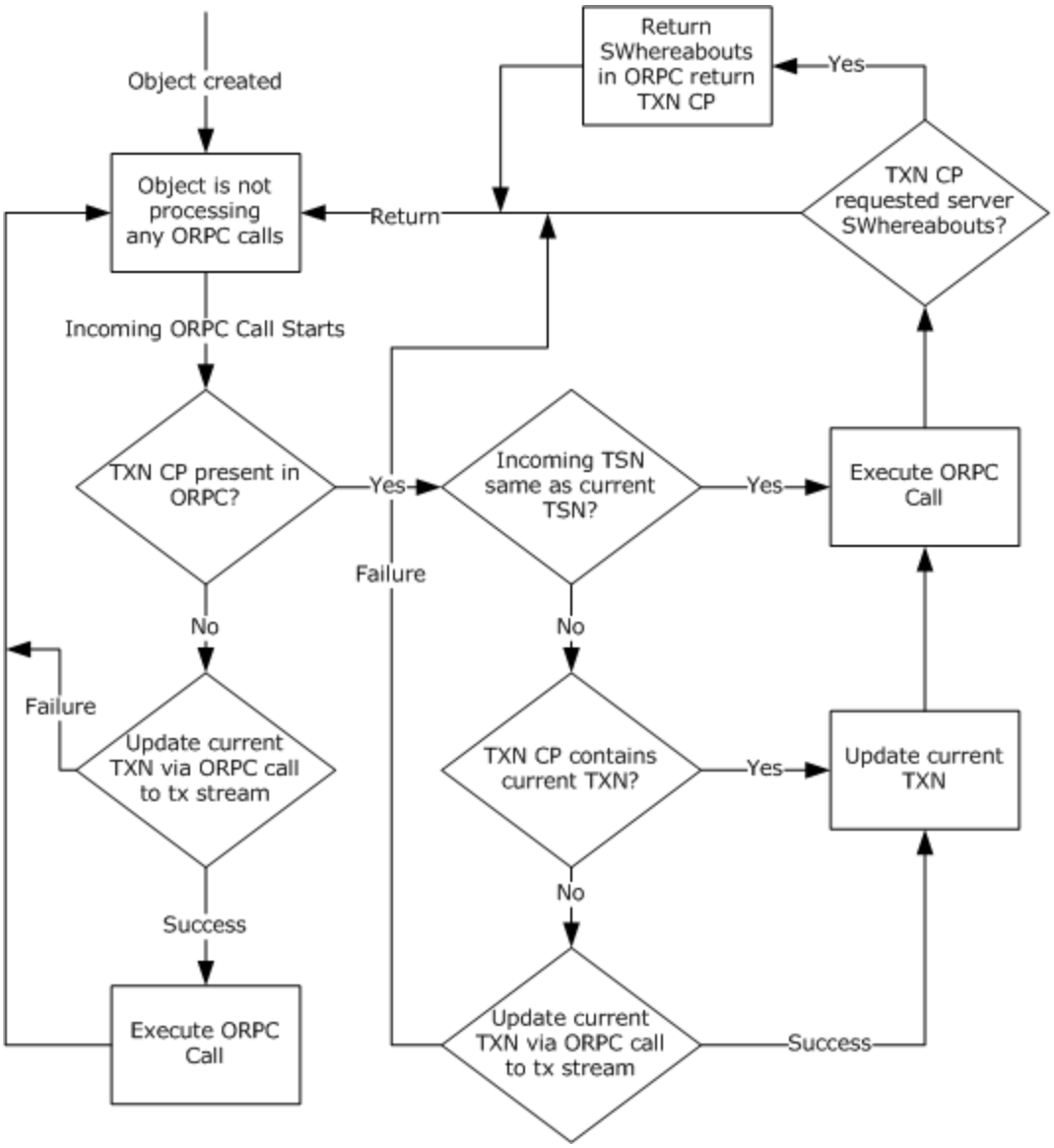


Figure 7: Incoming ORPC to a non-root transaction object

3.15.5 Timer Events

Not applicable to Server Transaction ORPC Extensions (section 3.15).

3.15.6 Other Local Events

Not applicable to Server Transaction ORPC Extensions (section 3.15).

3.16 Server Security ORPC Extension Details

3.16.1 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 explain 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.

The server security ORPC extension (section 3.16) MUST maintain a table of Security ORPC Extension (section 2.2.3.2.4) structures; each entry in the table MUST correspond to an incoming ORPC.

3.16.2 Timers

Not applicable to Server Security ORPC Extensions (section 3.16).

3.16.3 Initialization

Not applicable to Server Security ORPC Extensions (section 3.16).

3.16.4 Message Processing Events and Sequencing Rules

On receipt of an ORPC ([MS-DCOM] section 3.2.1.5.4), the Security ORPC Extension (section 2.2.3.2.4) MUST:

- Add an entry to the Security ORPC Extension table, and MUST set the fields in the entry to the corresponding fields in the Security ORPC Extension structure contained in the ORPC message.

When returning from an ORPC ([MS-DCOM] section 3.2.1.5.4), the Security ORPC Extension MUST remove the entry corresponding to the ORPC from the Security ORPC Extension table.

3.16.5 Timer Events

Not applicable to Server Security ORPC Extensions (section 3.16).

3.16.6 Other Local Events

Not applicable to Server Security ORPC Extensions (section 3.16).

3.17 Server Activity ORPC Processing Details

3.17.1 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 explain 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.

The server MUST maintain the data structures specified in section 3.9.1. In addition, the server MUST maintain:

- A table of objects, keyed by the activity ID.
- An activity timer for each activity.

- An activity lock.
- For each object, the causality identifier of the current outstanding ORPC call into the object, if any.

3.17.2 Timers

The server MUST maintain a per-activity timer.

3.17.3 Initialization

The server MUST initialize the per-activity timer to the time-out value of the activity set during the object activation request (see section 3.9.4).

3.17.4 Message Processing Events and Sequencing Rules

When processing an incoming ORPC call, the server MUST:

- Look up the activity ID of the object that is processing the ORPC call.
- Look up the list of objects that share the same activity.
- For each object in the list, the server MUST:
  - Determine the causality identifier of the outstanding ORPC call into the object.
  - If there is a causality identifier, the server MUST:
    - Compare the causality identifier of the outstanding ORPC call to that of the incoming ORPC call.
    - If they are the same, the server MUST allow the incoming ORPC call to the current object to be executed.
    - If they are different, the server MUST:
      - Start the activity timer.
      - Attempt to take the activity lock.
      - If the timer expires, the server MUST reject the incoming ORPC call as specified in section 3.17.5.
      - If the activity lock is acquired, the server MUST:
        - Set the causality identifier of the current object to that of the incoming ORPC call.
        - Reset the activity timer.
        - Allow the incoming ORPC call to be executed.
  - If there is no causality identifier, the server MUST examine the next object.
- If there are no outstanding ORPC calls on any object in the list, the server MUST:
  - Take the activity lock.
  - Set the causality identifier of the current object to that of the incoming ORPC call.

- Allow the incoming ORPC call to be executed.

- When the ORPC call completes, the server MUST:
  - Remove the causality identifier associated with the current object.
  - Relinquish the activity lock.

3.17.5 Timer Events

The server MUST return an error to each outstanding ORPC call when the activity timer of the current activity expires.

3.17.6 Other Local Events

Not applicable to Server Activity ORPC Processing (section 3.17).

3.18 Server Transaction Marshaling Details

3.18.1 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 explain 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.

The server transaction envoy (section 3.18) MUST maintain the following data structures:

- The transaction stream ID of the transaction stream used by the server object.
- A GUID that identifies uniquely the SWhereabouts obtained from the local implementation (as specified in [MS-DTCO]).

3.18.2 Timers

Not applicable to Server Transaction Envoy Marshaling (section 3.18).

3.18.3 Initialization

Not applicable to Server Transaction Envoy Marshaling (section 3.18).

3.18.4 Message Processing Events and Sequencing Rules

A transactional server object that is marshaled MUST produce an OBJREF_EXTENDED ([MS-DCOM] section 2.2.1.17.7) instance if the DCOM version of the client is 5.6 or greater. The transaction envoy MUST contribute a transaction envoy context property (see section 2.2.4.1) to the OBJREF_EXTENDED instance representing the marshaled object reference ([MS-DCOM] section 3.2.1.5.1). The transaction envoy property MUST be created as follows:

- The **TransactionStream** field MUST be set to the transaction stream ID of the transaction stream used by the server object.
- The **WhereaboutsID** field MUST be set to a GUID that identifies uniquely the SWhereabouts obtained from the local implementation (as specified in [MS-DTCO]).
- The **DtcCapabilities** field MUST be set to one or more of the values specified in section 2.2.6.1.

3.18.5 Timer Events

Not applicable to Server Transaction Envoy Marshaling (section 3.18).

3.18.6 Other Local Events

Not applicable to Server Transaction Envoy Marshaling (section 3.18).

3.19 Server Security Envoy Marshaling Details

3.19.1 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 explain 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.

The server security envoy (section 3.19) MUST maintain the following GUID data structure instances:

- domainGUID, which identifies the domain of the computer.
- machineGUID, which identifies the computer.

3.19.2 Timers

Not applicable to Server Security Envoy Marshaling (section 3.19).

3.19.3 Initialization

On initialization, the server security envoy (section 3.19) MUST:

- Set the domainGUID to the GUID of the domain of the computer. For more information, see [MS-NRPC] section 2.2.1.2.1.
- Set the machineGUID to a GUID that uniquely identifies the computer.

3.19.4 Message Processing Events and Sequencing Rules

When an object is marshaled, the server security envoy (section 3.19) MUST contribute a security envoy context property (section 2.2.4.2) to the OBJREF_EXTENDED ([MS-DCOM] section 2.2.1.17.7) instance representing the marshaled object reference ([MS-DCOM] section 3.2.1.5.1). The security envoy property MUST be created as follows:

- The **guidServerDomain** field MUST be set to domainGUID.
- The **guidServerMachine** field MUST be set to machineGUID.

3.19.5 Timer Events

Not applicable to Server Security Envoy Marshaling (section 3.19).

3.19.6 Other Local Events

Not applicable to Server Security Envoy Marshaling (section 3.19).

3.20 Client Transaction Unmarshaling Details

3.20.1 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 explain 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.

The client transaction envoy MUST maintain the following data structures, one for each OBJREF_EXTENDED object reference unmarshaled by the client:

- The transaction stream ID of the transaction stream used by the server object.
- A GUID that identifies uniquely the SWhereabouts obtained from the server object's implementation (as specified in [MS-DTCO]).
- A transaction sequence number (TSN) that identifies the last known good transaction acknowledged by the server object.
- One or more values from the **DtcCapabilities** enumeration (see section 2.2.6.1) identifying which protocols the server object's implementation supports.
- A global table that maps **WhereaboutsIDs** to cached copies of the server object's SWhereabouts (as specified in [MS-DTCO]).

3.20.2 Timers

Not applicable to Client Transaction Envoy Unmarshaling (section 3.20).

3.20.3 Initialization

Upon unmarshaling of a client transaction envoy, the client MUST:

- Compare the transaction stream ID of the unmarshaling client object to the transaction stream ID of the server object; if they do not match, the client transaction envoy is unnecessary, and the client MUST discard it.
- Otherwise, compare the capabilities of the local implementation to that of the server implementation, as returned in the **DtcCapabilities** field (see section 2.2.4.1); if the client and server do not share compatible implementations, then the client transaction envoy is unnecessary, and the client MUST discard it. For more information, see [MS-DTCO].
- Otherwise, the client MUST add the client transaction envoy to the client object's table of transaction envoy properties.

3.20.4 Message Processing Events and Sequencing Rules

3.20.5 Timer Events

Not applicable to Client Transaction Envoy Unmarshaling (section 3.20).

3.20.6 Other Local Events

When the client object releases its last reference to the server object, the client transaction envoy (section 3.20) MUST:

- Remove itself from the client object's table of transaction envoy properties.

3.21 Client Security Unmarshaling Details

3.21.1 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 explain 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.

The client security envoy MUST maintain the following GUID data structures, one for each OBJREF_EXTENDED object reference unmarshaled by the client:

- domainGUID, which identifies the domain of the computer of the object reference.
- machineGUID, which identifies the computer of the object reference.

3.21.2 Timers

Not applicable to Client Security Envoy Unmarshaling (section 3.21).

3.21.3 Initialization

Not applicable to Client Security Envoy Unmarshaling (section 3.21).

3.21.4 Message Processing Events and Sequencing Rules

When an OBJREF_EXTENDED ([MS-DCOM] section 2.2.1.17.7) object reference is unmarshaled ([MS-DCOM] section 3.1.4.1.2), the client security envoy MUST associate the following with the object reference:

- The **domainGUID** field MUST be set to guidServerDomain from the security envoy property.
- The **guidServerMachine** field MUST be set to guidServerMachine from the security envoy property.

3.21.5 Timer Events

Not applicable to Client Security Envoy Unmarshaling (section 3.21).

3.21.6 Other Local Events

Not applicable to Client Security Envoy Unmarshaling (section 3.21).

3.22 ITransactionStream Server Details

3.22.1 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 explain 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.

A transaction stream MUST maintain the following data structures:

- The currently active transaction sequence number (TSN).
- A Propagation_Token and/or STxInfo for the currently active transaction (as specified in [MS-DTCO]).
- A Boolean value denoting whether the current transaction MUST be canceled.

3.22.2 Timers

Not applicable to transaction streams.

3.22.3 Initialization

Upon creation, each transaction stream SHOULD:

- Initialize the TSN to a positive number.
- Initialize communication with its local transaction so that it is prepared to begin and complete transactions as needed.

3.22.4 Message Processing Events and Sequencing Rules

Methods in RPC Opnum Order

Method	Description
GetSeqAndTxViaExport	Returns an SWhereabouts and an STxInfo, and updates the client's currently known TSN. Opnum: 3
GetSeqAndTxViaTransmitter	Returns a Propagation_Token (as specified in [MS-DTCO]) and updates the client's currently known TSN. Opnum: 4
GetTxViaExport	Returns an STxInfo (as specified in [MS-DTCO]) for the requested TSN, or returns an error. Opnum: 5
GetTxViaTransmitter	Returns a Propagation_Token (as specified in [MS-DTCO]) for the specified TSN, or returns an error. Opnum: 6

The methods MUST NOT throw exceptions.

3.22.4.1 ITransactionStream::GetSeqAndTxViaExport (Opnum 3)

This method returns an SWhereabouts instance and an STxInfo instance, and updates the client's currently known TSN (as specified in [MS-DTCO]).

```
HRESULT GetSeqAndTxViaExport(
  [in] unsigned long ulKnownSeq,
  [in] unsigned long ulcbWhereabouts,
  [in, size_is(ulcbWhereabouts)]
    unsigned char* rgbWhereabouts,
  [out] unsigned long* pulCurrentSeq,
```

```
  [out] unsigned long* pulcbExportCookie,
  [out, size_is(,*pulcbExportCookie)]
    unsigned char** prgbExportCookie
);
```

ulKnownSeq: MUST specify the TSN for the current active transaction known by the client.

ulcbWhereabouts: MUST specify the size, in bytes, of *rgbWhereabouts*.

rgbWhereabouts: Contains an array of bytes that MUST contain the SWhereabouts of the client (as specified in [MS-DTCO]).

pulCurrentSeq: MUST specify the TSN for the currently active transaction.

pulcbExportCookie: MUST specify the size in bytes of *prgbExportCookie*.

prgbExportCookie: MUST contain an STxInfo for the currently active transaction (as specified in [MS-DTCO]).

Return Values: The method MUST return a positive value or zero, to indicate successful completion, or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

When processing this ORPC call, if *ulKnownSeq* is the same as the transaction stream's currently active TSN, the transaction stream MUST:

- Set *pulCurrentSeq* to *ulKnownSeq*.
- Set *pulcbExportCookie* to zero.
- Set *prgbExportCookie* to NULL.

Otherwise, the transaction stream MUST:

- Set *pulCurrentSeq* to *ulKnownSeq*.
- Obtain an STxInfo for the current active transaction (as specified in [MS-DTCO]).
- Set *pulcbExportCookie* to the size of the STxInfo (as specified in [MS-DTCO]).
- Copy the STxInfo to the *prgbExportCookie* out parameter (as specified in [MS-DTCO]).

3.22.4.2 ITransactionStream::GetSeqAndTxViaTransmitter (Opnum 4)

This method returns a Propagation_Token (as specified in [MS-DTCO]) and updates the client's currently known TSN.

```
HRESULT GetSeqAndTxViaTransmitter(
  [in] unsigned long ulKnownSeq,
  [out] unsigned long* pulCurrentSeq,
  [out] unsigned long* pulcbTransmitterBuffer,
  [out, size_is(,*pulcbTransmitterBuffer)]
    unsigned char** prgbTransmitterBuffer
);
```

ulKnownSeq: MUST specify the TSN for the current active transaction known by the client.

pulCurrentSeq: MUST specify the TSN for the currently active transaction.

pulcbTransmitterBuffer: MUST specify the size, in bytes, of prgbTransmitterBuffer.

prgbTransmitterBuffer: MUST contain a Propagation_Token for the currently active transaction (as specified in [MS-DTCO]).

Return Values: The method MUST return a positive value or zero, to indicate successful completion, or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

When processing this ORPC call, if *ulKnownSeq* is the same as the transaction stream's currently active TSN, the transaction stream MUST:

- Set *pulCurrentSeq* to *ulKnownSeq*.
- Set *pulcbTransmitterBuffer* to zero.
- Set *prgbTransmitterBuffer* to NULL.

Otherwise, the transaction stream MUST:

- Set *pulCurrentSeq* to *ulKnownSeq*.
- Obtain a Propagation_Token for the current active transaction (as specified in [MS-DTCO]).
- Set *pulcbTransmitterBuffer* to the size of the Propagation_Token (as specified in [MS-DTCO]).
- Copy the Propagation_Token to the *prgbTransmitterBuffer* out parameter (as specified in [MS-DTCO]).

3.22.4.3 ITransactionStream::GetTxViaExport (Opnum 5)

This method returns an STxInfo instance (as specified in [MS-DTCO]) for the requested TSN, or returns an error if the specified TSN is not the same as the transaction stream's currently active TSN.

```
HRESULT GetTxViaExport(
  [in] unsigned long ulRequestSeq,
  [in] unsigned long ulcbWhereabouts,
  [in, size_is(ulcbWhereabouts)]
    unsigned char* rgbWhereabouts,
  [out] unsigned long* pulcbExportCookie,
  [out, size_is(,*pulcbExportCookie)]
    unsigned char** prgbExportCookie
);
```

ulRequestSeq: MUST specify the TSN for the transaction being requested by the client.

ulcbWhereabouts: MUST specify the size, in bytes, of rgbWhereabouts.

rgbWhereabouts: Contains an array of bytes that MUST contain the SWhereabouts instance of the client (as specified in [MS-DTCO]).

pulcbExportCookie: MUST specify the size, in bytes, of *prgbExportCookie*.

prgbExportCookie: MUST contain an STxInfo instance for the currently active transaction (as specified in [MS-DTCO]).

Return Values: The method MUST return a positive value or zero to indicate successful completion, or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

When processing this ORPC call, if *ulRequestSeq* is the same as the transaction stream's currently active TSN, the transaction stream MUST:

- Obtain an STxInfo instance for the current active transaction.
- Set *pulcbExportCookie* to the size of the STxInfo instance(as specified in [MS-DTCO]).
- Copy the STxInfo instance to the *prgbExportCookie* out parameter (as specified in [MS-DTCO]).

Otherwise, the transaction stream MUST return an error.

3.22.4.4 ITransactionStream::GetTxViaTransmitter (Opnum 6)

This method returns a Propagation_Token (as specified in [MS-DTCO]) for the specified TSN, or returns an error if the specified TSN is not the same as the transaction stream's currently active TSN.

```
HRESULT GetTxViaTransmitter(
  [in] unsigned long ulRequestSeq,
  [out] unsigned long* pulcbTransmitterBuffer,
  [out, size_is(,*pulcbTransmitterBuffer)]
    unsigned char** prgbTransmitterBuffer
);
```

ulRequestSeq: MUST specify the TSN for the currently active transaction known by the client.

pulcbTransmitterBuffer: MUST specify the size, in bytes, of prgbTransmitterBuffer.

prgbTransmitterBuffer: MUST contain a Propagation_Token (as specified in [MS-DTCO]) for the currently active transaction.

Return Values: The method MUST return a positive value or zero to indicate successful completion, or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

When processing this ORPC call, if *ulRequestSeq* is the same as the transaction stream's currently active TSN, the transaction stream MUST:

- Obtain a Propagation_Token for the current active transaction from (as specified in [MS-DTCO]).
- Set *pulcbTransmitterBuffer* to the size of the Propagation_Token.
- Copy the Propagation_Token to the *prgbTransmitterBuffer* out parameter (as specified in [MS-DTCO]).

Otherwise, the transaction stream MUST return an error.

3.22.5 Timer Events

Not applicable to transaction streams.

3.22.6 Other Local Events

Not applicable to transaction streams.

4 Protocol Examples

The following section describes a commonly used operation to illustrate the function of the Component Object Model Plus (COM+) Protocol.

4.1 Client to RootTxn to NonrootTxn Example

Figure 8 shows a sequence for a client that activates and then calls a root txn object; the root txn activates and calls a non-root txn object, and returns a reference to the non-root txn object back to the client. The client then makes a call directly to the non-root txn object, then releases its references to both root and non-root objects.

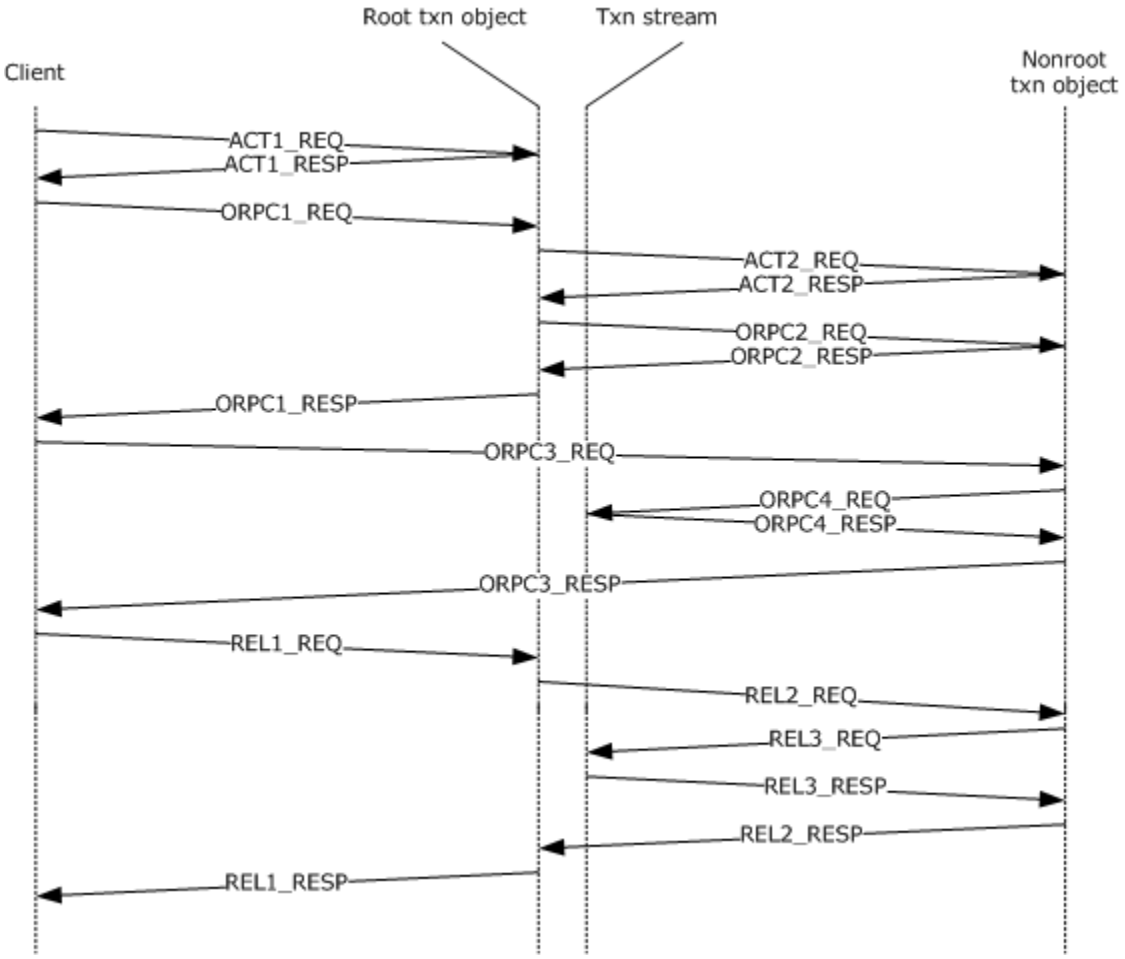


Figure 8: Client to RootTxn to NonrootTxn

ACT1_REQ: The client sends an activation request for the root txn object. The client is not an object, so the activation request does not contain any context properties.

During activation, the root txn object initializes itself and its associated transaction stream. In addition, the object is configured with an activity context property with an INFINITE activity time-out. The object is then marshaled as described in section 3.18, including a transaction envoy context property in the OBJREF_EXTENDED instance.

ACT1_RESP: The client receives the activation results, unmarshals the reference to the root object, and determines that the returned OBJREF contains a transaction envoy context property; however, since the client is not running within a transaction, it takes no special action.

ORPC1_REQ: The client makes a call to the root txn object. There are no ORPC extensions sent on this call, because the client previously discarded the transaction envoy property after unmarshaling the ACT1_RESP message.

At call entry to the root object, the root object's activity context property first checks to see that no other calls with different DCOM causality IDs are already executing in the object (see section 3.17). Since there are none, the call executes without delay.

ACT2_REQ: The root txn object activates the non-root txn object. As part of the activation, the root txn object sends a transaction context property (see section 2.2.2.1) that contains a reference to its associated transaction stream, and an activity context property (see section 2.2.2.2).

During activation, the non-root txn object is created and initialized using the same transaction and the same activity as the root txn object. The non-root txn object saves a reference to the root object's transaction stream; marshals itself, as described in section 3.18, including a transaction envoy context property in the OBJREF_EXTENDED instance; and configures its server activity context property using the activity context property supplied during the activation. The non-root object is then marshaled as described in section 3.18, including a transaction envoy context property in the OBJREF_EXTENDED.

ACT2_RESP: The root txn object unmarshals the reference to the non-root txn object; detects the returned transaction envoy context property; notes that both it and the non-root txn object are running in the same transaction; and adds the returned envoy property to its table for action on subsequent calls.

ORPC2_REQ: The root txn object makes a call to the non-root txn object. Since the root object has a transaction envoy property for this reference, it includes a Transaction ORPC extension on the call as described in section 2.2.3.1.1.

The non-root txn object receives the call and goes through the steps described in section 3.15 to ensure that it is running within a valid transaction. In addition, at call entry to the non-root object, the root object's activity context property first checks to see that no other calls with different DCOM causality IDs are already executing in the object (see section 3.17). Since there are none, the call executes without delay.

ORPC2_RESP: The non-root txn object returns to the root txn object. This response carries a Transaction ORPC extension on the return, as described in section 2.2.3.1.2.

ORPC1_RESP: The root txn object returns to the client. One of the return parameters contains a reference to the non-root txn object, so that the client can call the non-root txn object directly in future.

The client unmarshals the reference to the non-root txn object. As in ACT1_RESP, the client ignores the transaction envoy context property coming from the non-root txn object.

ORPC3_REQ: The client makes an ORPC call directly to the non-root txn object. There are no ORPC extensions sent on this call.

ORPC4_REQ: The non-root txn object receives the call from the client and determines that no Transaction ORPC Call Extension is present. To ensure that it is running within a valid transaction, it makes an ORPC call to the root object's associated transaction stream, using either the *GetSeqAndTxViaExport* (see section 3.22.4.1) or the *GetSeqAndTxViaTransmitter* (see section 3.22.4.2) method.

ORPC4_RESP: The transaction stream determines that the TSN specified by the non-root txn object is still valid and returns a success code.

ORPC3_RESP: The non-root txn object returns to the client. Since ORPC3_REQ did not have a Transaction ORPC Call extension, the response also does not have a Transaction ORPC Return extension.

REL1_REQ: The client releases its reference to the root txn object. Since this is the last reference on the root txn object, it is destroyed.

REL2_REQ: The root txn object releases its reference on the non-root txn object.

REL3_REQ: The non-root txn object releases its references on the transaction stream.

REL3_RESP: Returns from the release call on the transaction stream.

REL2_RESP: Returns from the release call on the non-root txn object.

REL1_RESP: Returns from the release call on the root txn object. Because this is the last release on the root txn object, the transaction is committed at this point before returning.

5 Security

The following sections specify security considerations for implementers of the Component Object Model Plus (COM+) Protocol.

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided, where "ms-dcom.idl" is the IDL found in [MS-DCOM] Appendix A.

```
import "ms-dcom.idl";
// Disable new Vista MIDL attribute if using an older MIDL compiler
#if __midl < 700
#define disable_consistency_check
#endif
[
    object,
    uuid(97199110-DB2E-11d1-A251-0000F805CA53),
    pointer_default(unique)
]
interface ITransactionStream : IUnknown
{
    HRESULT GetSeqAndTxViaExport (
        [in] unsigned long ulKnownSeq,
        [in] unsigned long ulcbWhereabouts,
        [in, size_is(ulcbWhereabouts)] unsigned char* rgbWhereabouts,
        [out] unsigned long* pulCurrentSeq,
        [out] unsigned long* pulcbExportCookie,
        [out, size_is(,*pulcbExportCookie)]
        unsigned char** prgbExportCookie);

    HRESULT GetSeqAndTxViaTransmitter (
        [in] unsigned long ulKnownSeq,
        [out] unsigned long* pulCurrentSeq,
        [out] unsigned long* pulcbTransmitterBuffer,
        [out, size_is(,*pulcbTransmitterBuffer)]
        unsigned char** prgbTransmitterBuffer);

    HRESULT GetTxViaExport (
        [in] unsigned long ulRequestSeq,
        [in] unsigned long ulcbWhereabouts,
        [in, size_is(ulcbWhereabouts)] unsigned char* rgbWhereabouts,
        [out] unsigned long* pulcbExportCookie,
        [out, size_is(,*pulcbExportCookie)]
        unsigned char** prgbExportCookie);

    HRESULT GetTxViaTransmitter (
        [in] unsigned long ulRequestSeq,
        [out] unsigned long* pulcbTransmitterBuffer,
        [out, size_is(,*pulcbTransmitterBuffer)]
        unsigned char** prgbTransmitterBuffer);
};
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista

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 2.2.2.3.1: For historical reasons, Windows will set this field to random values.

<2> Section 3.9.4: Windows uses an activity time-out of INFINITE by default.

<3> Section 3.12.4: Windows servers will set this field to the NETBIOS name of the server machine.

<4> Section 3.12.4: Windows servers will set this field to the NETBIOS name and IPV4 addresses of the server machine.

<5> Section 3.12.4: Windows servers will set this field to the IPV6 addresses of the server machine.

8 Index

A

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W