[MS-CMPO]:
MSDTC Connection Manager:
OleTx Transports Protocol Specification

Intellectual Property Rights Notice for Protocol Documentation

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

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
04/03/2007	0.01		MCPP Milestone Longhorn Initial Availability
07/03/2007	1.0	Major	MLonghorn+90
07/20/2007	1.1	Minor	Updated the technical content.
08/10/2007	1.1.1	Editorial	Revised and edited the technical content.

Copyright © 2008 Microsoft Corporation.

Date	Revision History	Revision Class	Comments
09/28/2007	2.0	Major	Made a change to the IDL.
10/23/2007	3.0	Major	Updated and revised the technical content.
11/30/2007	3.0.1	Editorial	Revised and edited the technical content.
01/25/2008	3.0.2	Editorial	Revised and edited the technical content.

Copyright © 2008 Microsoft Corporation.

Table of Contents

Copyright © 2008 Microsoft Corporation.

Copyright © 2008 Microsoft Corporation.

1 Introduction

This document specifies the MSDTC Connection Manager: OleTx Transports Protocol. The MSDTC Connection Manager: OleTx Transports Protocol is a **remote procedure call (RPC)** interface for establishing duplex **sessions** between two **partners** and for exchanging messages between them. The MSDTC Connection Manager: OleTx Transports Protocol is a framing and message transport protocol and, as such, is designed to have other protocols layered over the basic session, messaging, and security services that it provides.

1.1 Glossary

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

64-Bit Network Data Representation (NDR64)
Authentication Level
Authentication Service (AS)
Client
Dynamic Endpoint
Endpoint
Endpoint Mapper
Globally Unique Identifier (GUID)
Interface Definition Language (IDL)
Microsoft Interface Definition Language (MIDL)
Network Data Representation (NDR)
Opnum
Remote Procedure Call (RPC)
RPC Protocol Sequence
RPC Server
RPC Transfer Syntax
RPC Transport
Security Provider
Universally Unique Identifier (UUID)

The following terms are specific to this document:

Contact Identifier (CID): A **universally unique identifier (UUID)** that identifies a **partner** in the MSDTC Connection Manager: OleTx Transports Protocol. These **UUIDs** are frequently converted to and from string representations. This string representation MUST follow the format specified in [C706] Appendix A. In addition, the **UUIDs** MUST be compared, as specified in [C706] Appendix A.

Level-two Protocol: The MSDTC Connection Manager: OleTx Transports Protocol is designed to be a transport protocol over which two other protocols are layered. When used in this document, level-two protocol refers to the protocol that is layered immediately on top of MSDTC Connection Manager: OleTx Transports Protocol, as described in 2.2.4.

Level-three Protocol: The MSDTC Connection Manager: OleTx Transports Protocol is designed to be a transport protocol over which two other protocols are layered. When used in this document, level-three protocol refers to the protocol that is layered immediately on top of the **level-two protocol**, as described in 2.2.4.

Partner: A participant in the MSDTC Connection Manager: OleTx Transports Protocol. Each **partner** has its own **contact identifier (CID)**, and uses the IXnRemote interface to invoke and receive **remote procedure calls (RPCs)**.

Copyright © 2008 Microsoft Corporation.

Primary Partner: One of the two participants in an MSDTC Connection Manager: OleTx Transports Protocol **session**. The **primary partner** is the **partner** with the larger **contact identifier (CID)**, as specified in [C706] Appendix A.

Secondary Partner: One of the two participants in an MSDTC Connection Manager: OleTx Transports Protocol **session**; the **secondary partner** is the **partner** with the smaller **contact identifier (CID)**, as specified in [C706] Appendix A.

Session: An association of two MSDTC Connection Manager: OleTx Transports Protocol **partners** that want to exchange messages.

Session Rank: The role of a **partner** in an MSDTC Connection Manager: OleTx Transports Protocol **session**, either primary or secondary. The rank is determined by comparing the **contact identifiers (CIDs)** of the two **partners** (as specified in [C706] Appendix A); the **partner** with the larger **contact identifier (CID)** is the **primary partner**, and the other is the **secondary partner**.

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

1.2 References

1.2.1 Normative References

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

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

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

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

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

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[NETBEUI] IBM Corporation, "LAN Technical Reference: 802.2 and NetBIOS APIs", 1986, http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/BK8P7001/CCONTENTS

If you have any trouble finding [NETBEUI], please check here.

[RFC1001] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", RFC 1001, March 1987, http://www.ietf.org/rfc/rfc1001.txt

[RFC1002] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications", RFC 1002, March 1987, http://www.ietf.org/rfc/rfc1002.txt

[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

Copyright © 2008 Microsoft Corporation.

1.2.2 Informative References

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

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

[MS-SPNG] Microsoft Corporation, "Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions", January 2007.

1.3 Protocol Overview (Synopsis)

The MSDTC Connection Manager: OleTx Transports Protocol is a peer-to-peer messaging protocol layered over a bi-directional pair of RPC connections. Although there is asymmetry in the setup and teardown of a session, the peers (or partners) are considered equal for the purposes of sending messages to each other.

Together, the pair of RPC connections between the partners is called a session.

1.3.1 Identifiers and Partner Roles

Each of the partners involved in an MSDTC Connection Manager: OleTx Transports Protocol session has a distinct **universally unique identifier (UUID)** called its **contact identifier (CID)**. Each partner is identified by the combination of its CID and the NetBIOS name of the computer in which it resides. For more information on NetBIOS, see [NETBEUI], [RFC1001], and [RFC1002].

There are two slightly different roles in the MSDTC Connection Manager: OleTx Transports Protocol: **primary partner** and **secondary partner**. Any partner has the option to take either role, but within a session, one is chosen to be the primary partner, and the other is chosen to be the secondary partner. (A partner's role in the session is also referred to as its **session rank**.) Each partner in the pair self-determines its role by comparing its CID with the CID of the other partner. For comparing UUID, see [C706]. The partner that has the larger CID is the primary partner; the other partner is the secondary partner.

1.3.2 Finding the RPC Endpoint and Constructing a Binding Handle

When a partner is initialized, it creates a **dynamic endpoint** on each of its supported RPC protocols and registers the interface (IXnRemote) with the RPC **endpoint mapper**. When a partner performs this registration, it specifies its contact identifier (CID) as the object identifier. See specification [C706].

A partner initiating communication with another partner begins with a name object that contains contact information for a remote partner. The name object is used to create an RPC binding handle (see specification [C706]) to the remote partner's RPC **endpoint**. The name object contributes to the binding as follows:

1. The binding's protocol sequence is taken from one of the entries in the Protocols list in the name object. The protocol MUST be one of the protocols supported by both partners and is selected as specified in section 2.1.1.

2. The binding's host network address is specified as the Hostname in the name object.

3. The binding's object UUID is specified as the contact identifier (CID) in the name object.

Copyright © 2008 Microsoft Corporation.

This partial binding is resolved into a full binding by using the RPC endpoint mapper service at the host network address and the full binding handle is used for every call to the remote partner.

1.3.3 Session Lifecycle

The following sections specify supported MSDTC Connection Manager: OleTx Transports Protocol **RPC protocol sequences** for implementers.

1.3.3.1 Establishing a Session

A session is established by making a nested series of synchronous remote procedure call (RPC) between the IXnRemote interfaces of the two partners. These calls are made in order; furthermore, no call begins before the last call completes, unless an error occurs.

Once one of the partners decides to establish a session, the sequence is as follows. If the primary partner decides to establish the session, it proceeds immediately. If the secondary partner decides to establish the session, it establishes an RPC connection to the primary partner and calls either the **Poke** method or the **PokeW** method, which has the effect of informing the primary that the secondary wants to establish a session. The primary partner begins the handshake series by establishing an RPC connection to the secondary partner, and by making a **BuildContext** call or **BuildContextW** call to the secondary partner. The secondary partner responds to the incoming call by making a corresponding **BuildContext** callback or **BuildContextW** callback to the primary (after establishing an RPC connection, if necessary).

The primary partner then verifies the callback, and the chain of procedure calls begins to unwind. The primary partner returns from the **BuildContext** call or the **BuildContextW** call that was made by the secondary partner, and then the secondary partner returns from the **BuildContext** call or the **BuildContextW** call that was made by the primary. Once these calls have returned, the session has been established. The following sequence diagrams illustrate this process.

Copyright © 2008 Microsoft Corporation.

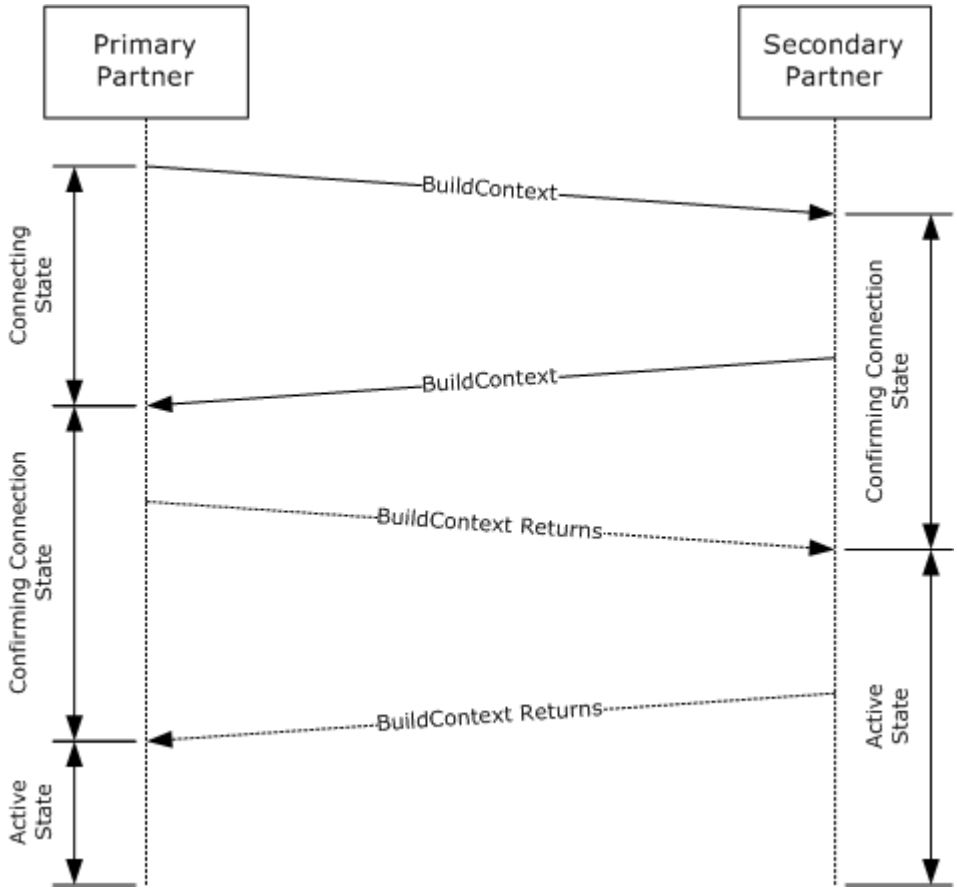


Figure 1: Session initiation by primary

Copyright © 2008 Microsoft Corporation.

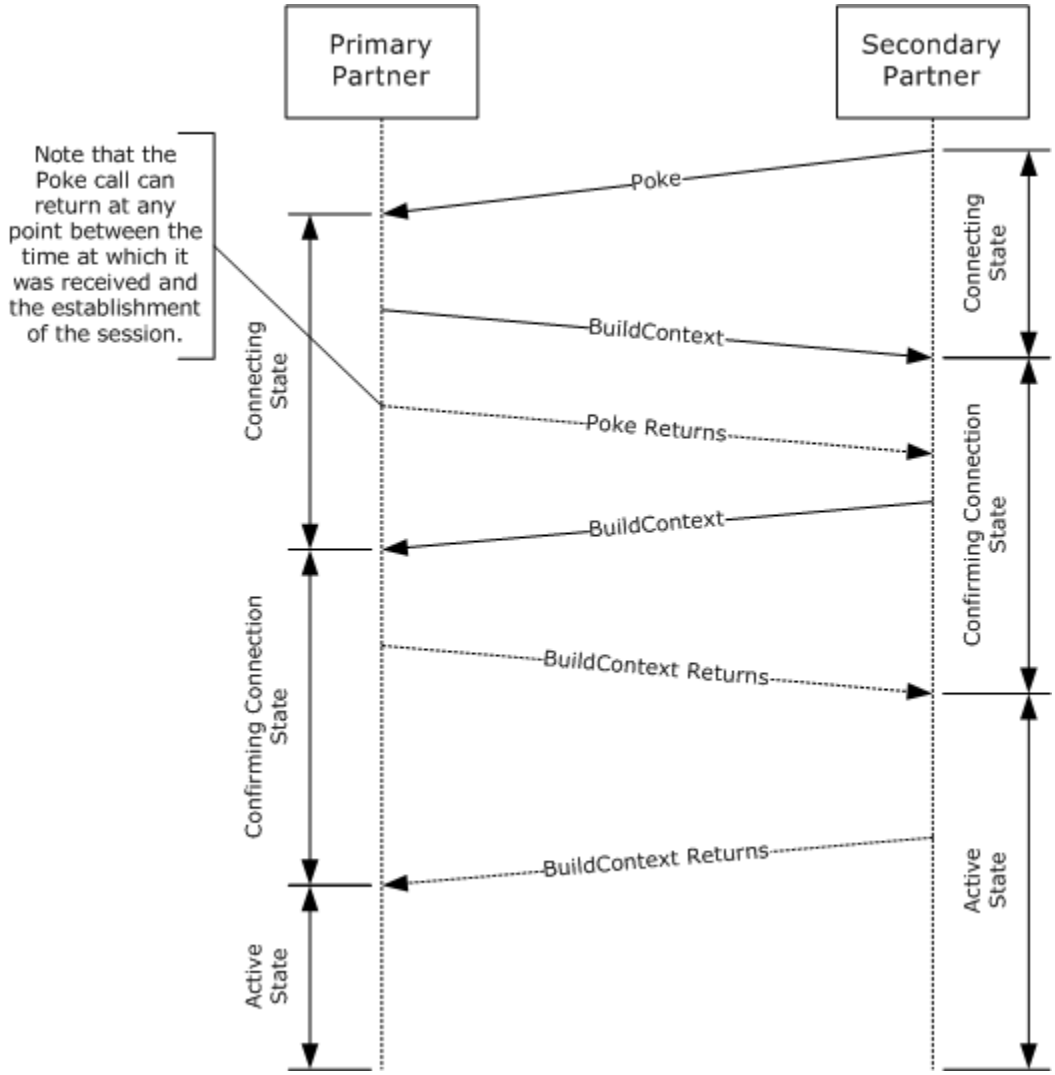


Figure 2: Session initiation by secondary

1.3.3.2 Negotiating Resources

Once a session has been established, a partner has the option to call the **NegotiateResources** method to request that the other partner allocate resources to be associated with the session. The MSDTC Connection Manager: OleTx Transports Protocol does not define any resource types or requirements on its own; the definition of such things is left to the particular protocol being layered over it.

1.3.3.3 Sending and Receiving Messages

Once a session has been established, a partner calls the **SendReceive** method to send messages to the other partner. As with resources, the MSDTC Connection Manager: OleTx Transports Protocol does not define any messages or message formats; the definition of such things is left to the particular protocol being layered over it.

Copyright © 2008 Microsoft Corporation.

1.3.3.4 Terminating a Session

Termination requires a nested series of remote procedure calls (RPCs) between the IXnRemote interfaces of the two partners. Either partner has the option to terminate the session. If the primary partner decides to terminate the session, it proceeds immediately. If the secondary partner decides to terminate the session, it sends a **BeginTearDown** request to the primary partner, which has the effect of informing the primary to terminate the session.

The primary partner begins the handshake series by making a **TearDownContext** call to the secondary partner. The secondary partner responds by freeing some of its local state and making a corresponding **TearDownContext** callback to the primary partner.

On receiving this callback, the primary partner frees its local state associated with the session.

Note that the exact conditions under which a partner decides to terminate a session are outside the scope of the MSDTC Connection Manager: OleTx Transports Protocol; it is the responsibility of the protocol being layered above the MSDTC Connection Manager: OleTx Transports Protocol to provide mechanisms for determining the lifetime of a session.

1.4 Relationship to Other Protocols

The MSDTC Connection Manager: OleTx Transports Protocol is dependent on RPC, which is its transport. The protocol provides extensibility elements that are used by the MSDTC Connection Manager: OleTx Transports Protocol to provide connection multiplexing and other services. Other message-based protocols, such as [MS-DTCO], are layered on top of that to provide application-specific functionality.

Ultimately, this suite of protocols is used as the communication mechanism for the Microsoft Distributed Transaction Coordinator, which is used to coordinate atomic transactions.

1.5 Prerequisites/Preconditions

The MSDTC Connection Manager: OleTx Transports Protocol is an RPC interface, and therefore has the prerequisites identified in [MS-RPCE] as being common to RPC interfaces.

The security model employed by this protocol is based on the **Security Provider** model specified in [MS-RPCE], section 1.7. As a result, the function of the protocol requires the availability of a Security Provider infrastructure that can be used for RPC security.

It is assumed that an MSDTC Connection Manager: OleTx Transports Protocol partner has obtained the contact information for another partner that supports the MSDTC Connection Manager: OleTx Transports Protocol before establishing a session. How a partner does this is not addressed in this specification.

1.6 Applicability Statement

The MSDTC Connection Manager: OleTx Transports Protocol is primarily designed to provide a peer-to-peer system for exchanging messages over reliable connections. Its use of bi-directional RPC connections to RPC dynamic endpoints means that it is only applicable when the participants can directly contact each other. The MSDTC Connection Manager: OleTx Transports Protocol requires that the partners refer to each other by NetBIOS name; that is, the participants should use a name service. Also, the use of Mutual Authentication in conjunction with the protocol's reliance on NetBIOS means that the participants are required to be either in the same domain or in domains that have a trust relationship.

Copyright © 2008 Microsoft Corporation.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- Supported **RPC Transports**: The MSDTC Connection Manager: OleTx Transports Protocol uses multiple RPC protocol sequences, as specified in section 1.3.3.
- Protocol Versions: The MSDTC Connection Manager: OleTx Transports Protocol RPC interface has a single version number of 1.0; however, there are two instances of this interface:
  - A base interface.
  - An extended interface obtained by appending methods at the end of the base interface described in section 3.1.

  Corresponding to the two interface instances, this protocol defines two versions, which for the purposes of this specification are referred as "MS-CMPO 1.0" (implements the base interface) and "MS-CMPO 1.1" (implements the extended interface).

  It is possible to further extend the MSDTC Connection Manager: OleTx Transports Protocol without altering the interface version number by adding RPC methods to the interface with **opnums** lying numerically beyond those defined in this specification.

  A **client** determines support for a certain interface instance (or protocol version) from a server by attempting to invoke an instance-specific method. If the method is not supported, the **RPC Server** MUST return an RPC_S_PROCNUM_OUT_OF_RANGE error. For RPC versioning and capacity negotiation in this situation, see [C706], section 4.2.4.2, and [MS-RPCE], section 1.7.
- Security and Authentication Methods: When using authentication, the MSDTC Connection Manager: OleTx Transports Protocol uses the Security Provider security model as specified in [MS-RPCE], section 3.1.1.5.4. The specific methods of authentication for this protocol are highly implementation-dependent. In order to communicate securely, two protocol partners MUST agree on a common Security Provider package to use. Security Provider negotiation packages are specified in [MS-SPNG].<1>

1.8 Vendor-Extensible Fields

The MSDTC Connection Manager: OleTx Transports Protocol uses **HRESULTs**, as specified in [MS-ERREF]. Vendors are free to choose their own values for this field, as long as the C bit (0x20000000) is set, indicating it is a customer code.

1.9 Standards Assignments

Parameter	Value	Reference
RPC interface UUID	906B0CE0-C70B-1067-B317-00DD010662DA	Section 2.1

Copyright © 2008 Microsoft Corporation.

2 Messages

The following sections specify how MSDTC Connection Manager: OleTx Transports Protocol messages are encapsulated on the wire and common MSDTC Connection Manager: OleTx Transports Protocol data types.

2.1 Transport

2.1.1 Protocol Sequences

The MSDTC Connection Manager: OleTx Transports Protocol uses several different RPC protocol sequences; it SHOULD use the "ncacn_ip_tcp" RPC protocol sequence.

Also, the MSDTC Connection Manager: OleTx Transports Protocol uses several different RPC protocol sequences, and MAY use any or all of the "ncacn_ip_udp", "ncacn_nb_nb", and "ncacn_spx" sequences. Very few implementations use these protocols, and so they SHOULD NOT be the only protocols supported by a partner.<2>

2.1.2 Endpoints

The MSDTC Connection Manager: OleTx Transports Protocol uses RPC dynamic endpoints, as specified in [C706] part 4.

2.1.3 Security

The MSDTC Connection Manager: OleTx Transports Protocol SHOULD use Security Provider, as specified in [MS-RPCE].<3>

2.2 Common Data Types

The MSDTC Connection Manager: OleTx Transports Protocol MUST indicate (to the RPC runtime) that it is only to support the **Network Data Representation (NDR)** transfer syntax as the **RPC transfer syntax**, as specified in [C706] part 4. In addition to RPC base types and definitions specified in [C706] and [MS-DTYP], additional data types are defined in the following sections.

2.2.1 HRESULT

This specification uses the HRESULT type. See [MS-ERREF].

2.2.2 TEARDOWN_TYPE

The **TEARDOWN_TYPE** enumeration provides a set of enumerated values indicating the reason for starting the teardown phase of session management.

```
typedef  enum _TearDownType
{
  TT_FORCE = 0,
  TT_PROBLEM = 2
} TEARDOWN_TYPE;
```

TT_FORCE: Force a teardown.

Copyright © 2008 Microsoft Corporation.

TT_PROBLEM: Severe session error detected; start a teardown.

2.2.3 SESSION_RANK

The **SESSION_RANK** enumeration provides values that describe whether the machine is a primary partner or a secondary partner.

```
typedef  enum _SessionRank
{
  SRANK_PRIMARY = 1,
  SRANK_SECONDARY = 2
} SESSION_RANK;
```

SRANK_PRIMARY: Primary partner.

SRANK_SECONDARY: Secondary partner.

2.2.4 BIND_VERSION_SET

The **BIND_VERSION_SET** structure holds three sets of version range values that specify the version ranges supported by a partner for three protocols: this protocol, MSDTC Connection Manager: OleTx Transports Protocol, and two other protocols that are layered on top of this protocol.This is because MSDTC Connection Manager: OleTx Transports Protocol is designed to be a transport protocol over which two other protocols are layered. For the rest of this specification, we refer to the protocol that is layered immediately on top of MSDTC Connection Manager: OleTx Transports Protocol as the **level-two protocol**, and to the protocol layered on top of the level-two protocol as the **level-three protocol**. The ranges of level-two version number values and level-three version number values are specific to the level-two protocol and level-three protocol, respectively.

```
typedef struct _BindVersionSet {
  DWORD dwMinLevelOne;
  DWORD dwMaxLevelOne;
  DWORD dwMinLevelTwo;
  DWORD dwMaxLevelTwo;
  DWORD dwMinLevelThree;
  DWORD dwMaxLevelThree;
} BIND_VERSION_SET;
```

dwMinLevelOne: A 4-byte unsigned integer value containing the minimum supported MSDTC Connection Manager: OleTx Transports Protocol version. **dwMinLevelOne** MUST be less than or equal to **dwMaxLevelOne**.

dwMaxLevelOne: A 4-byte unsigned integer value containing the maximum version supported for a level-one session. **dwMaxLevelOne** MUST be greater than or equal to **dwMinLevelOne**.

This field MUST be one of the following values:

Value	Meaning
0x00000001	The ANSI character set should be used.

Copyright © 2008 Microsoft Corporation.

Value	Meaning
0x00000002	The Unicode character set should be used.

dwMinLevelTwo: A 4-byte unsigned integer value containing the minimum version supported for the level-two protocol session. The value for **dwMinLevelTwo** MUST be less than or equal to **dwMaxLevelTwo**.

dwMaxLevelTwo: A 4-byte unsigned integer value containing the maximum version supported for the level-two protocol session. The value for **dwMaxLevelTwo** MUST be greater than or equal to **dwMinLevelTwo**.

dwMinLevelThree: A 4-byte unsigned integer value containing the minimum version supported for the level-three protocol session. The value for **dwMinLevelThree** MUST be less than or equal to **dwMaxLevelThree**.

dwMaxLevelThree: A 4-byte unsigned integer value containing the maximum version supported for the level-three protocol session. **dwMaxLevelThree** MUST be greater than or equal to **dwMinLevelThree**.

2.2.5 BOUND_VERSION_SET

The **BOUND_VERSION_SET** is a structure containing the MSDTC Connection Manager: OleTx Transports Protocol version numbers that were successfully negotiated during a **BuildContext** call or a **BuildContextW** call.

```
typedef struct _BoundVersionSet {
  DWORD dwLevelOneAccepted;
  DWORD dwLevelTwoAccepted;
  DWORD dwLevelThreeAccepted;
} BOUND_VERSION_SET;
```

dwLevelOneAccepted: A session level-one bind was successfully created.

A 4-byte unsigned integer value containing the MSDTC Connection Manager: OleTx Transports Protocol version that was negotiated with the partner and MUST be used in MSDTC Connection Manager: OleTx Transports Protocol protocol exchanges with the partner.

Value	Meaning
0x00000001	ANSI character set should be used.
0x00000002	Unicode character set should be used.

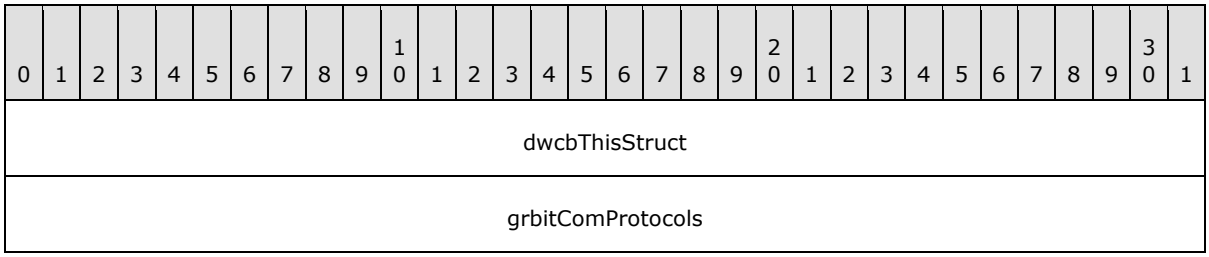
dwLevelTwoAccepted: A 4-byte unsigned integer value containing the level-two protocol version that was negotiated with the partner and MUST be used in level-two protocol exchanges with the partner.

dwLevelThreeAccepted: A 4-byte unsigned integer value containing the level-three protocol version that was negotiated with the partner and MUST be used in level-three protocol exchanges with the partner.

Copyright © 2008 Microsoft Corporation.

2.2.6 BIND_INFO_BLOB

The BIND_INFO_BLOB packet is a custom-marshaled structure containing details on how to bind to a partner.

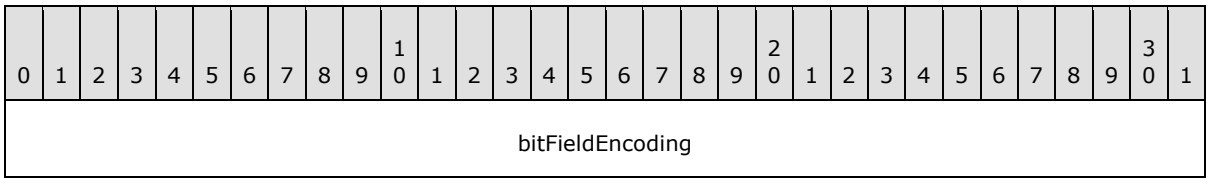


dwcbThisStruct (4 bytes): An unsigned 4-byte integer. The size of this structure in bytes. This value MUST be set to 8.

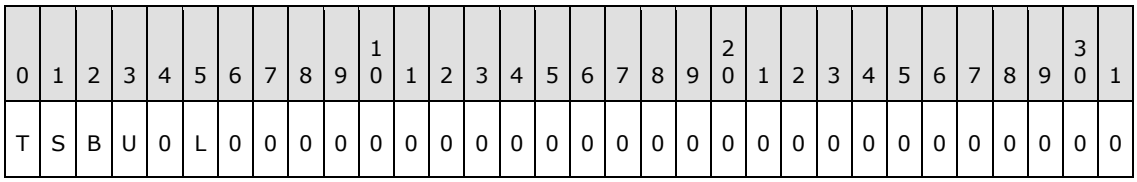
grbitComProtocols (4 bytes): A COM_PROTOCOL bit field specifying what RPC protocol sequences the partner supports.

2.2.7 COM_PROTOCOL

The COM_PROTOCOL is a bit field defining the set of RPC protocol sequences supported by an MSDTC Connection Manager: OleTx Transports Protocol partner.



bitFieldEncoding (4 bytes): The bits of this data type are encoded as follows:



Value	Description
T	A flag indicating whether or not the "ncacn_ip_tcp" RPC protocol sequence is supported by the endpoint. If the value is 1, the protocol sequence is supported; otherwise, it is not.
S	A flag indicating whether or not the "ncacn_spx" RPC protocol sequence is supported by the endpoint. If the value is 1, the protocol sequence is supported; otherwise, it is not.
B	A flag indicating whether or not the "ncacn_nb_nb" RPC protocol sequence is supported by the endpoint. If the value is 1, the protocol sequence is supported; otherwise, it is not.
U	A flag indicating whether or not the "ncacn_ip_udp" RPC protocol sequence is supported by the endpoint. If the value is 1, the protocol sequence is supported; otherwise, it is not.

Copyright © 2008 Microsoft Corporation.

Value	Description
L	A flag used to indicate implementation-dependent support for local communications.<4>

Copyright © 2008 Microsoft Corporation.

3 Protocol Details

The RPC interface specified by this protocol is called IXnRemote (see section 6 for the **IDL** specification). Every IXnRemote client is also an IXnRemote server, and every IXnRemote server is also an IXnRemote client. Therefore, the information in sections 3.2 and 3.3 apply equally to all participants in the MSDTC Connection Manager: OleTx Transports Protocol.

3.1 Protocol Versioning

This protocol currently has two versions: MS-CMPO 1.0 and MS-CMPO 1.1. The only differences between the two versions are related to the methods supported by the RPC interface as shown in the following table:

IXnRemote methods	MS-CMPO 1.0	MS-CMPO 1.1
Poke (Opnum 0)	Supported	Supported
BuildContext (Opnum 1)	Supported	Supported
NegotiateResources (Opnum 2)	Supported	Supported
SendReceive (Opnum 3)	Supported	Supported
TearDownContext (Opnum 4)	Supported	Supported
BeginTearDown (Opnum 5)	Supported	Supported
PokeW (Opnum 6)	Not supported	Supported
BuildContextW (Opnum 7)	Not supported	Supported

3.2 IXnRemote Server 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 what is described in this document.

3.2.1.1 Local Partner State

An MSDTC Connection Manager: OleTx Transports Protocol partner MUST maintain the following local data elements:

Local Name Object: A name object that contains the contact information for the partner.

Minimum Level 1 Version Number: A 4-byte unsigned integer, whose value represents the minimum version supported by a MSDTC Connection Manager: OleTx Transports Protocol implementation.

Maximum Level 1 Version Number: A 4-byte unsigned integer, whose value represents the maximum version supported by a MSDTC Connection Manager: OleTx Transports Protocol implementation.

Copyright © 2008 Microsoft Corporation.

Minimum Level 2 Version Number: A 4-byte unsigned integer, whose value represents the minimum version supported by the level-two protocol layered on top of the MSDTC Connection Manager: OleTx Transports Protocol implementation.

Maximum Level 2 Version Number: A 4-byte unsigned integer, whose value represents the maximum version supported by the level-two protocol layered on top of the MSDTC Connection Manager: OleTx Transports Protocol implementation.

Minimum Level 3 Version Number: A 4-byte unsigned integer, whose value represents the minimum version supported by the level-three protocol layered on top of the level-two protocol.

Maximum Level 3 Version Number: A 4-byte unsigned integer, whose value represents the maximum version supported by the level-three protocol layered on top of the level-two protocol.

Server Security Settings: An implementation-specific object, which contains Security Provider-specific settings that are used in setting up the RPC security of the server.

Client Security Settings: An implementation-specific object, which contains Security Provider-specific settings that are used in setting up the RPC security of the client.

Security Level: An implementation-specific enumeration value which specifies the security behavior of a protocol partner. The generic values of this enumeration are given in the following table:

Security Level value	Meaning
Mutual Authentication	This value specifies that the protocol partner MUST use secured RPC communication for both client and server. The server RPC security MUST be configured as specified by the Server Security Settings, and the client security MUST be configured as specified by the Client Security Settings.
Incoming Authentication	This value specifies that the protocol partner MUST use secured RPC communication for sessions that were initiated (through **BuildContextW** or **PokeW**) by another protocol partner. For sessions initiated by itself, a partner MUST first attempt to use secured RPC communication; if that is not supported, it MUST use unsecured RPC communication.
Fallback to Unsecured if Necessary	This value specifies that both the client and server sides of the protocol partner MUST fallback to unsecured RPC if secured RPC is not supported.
Unsecured	This value specifies that both the client and server sides of the protocol partner MUST use unsecured RPC communication. The settings specified by the Server Security Settings and Client Security Settings objects MUST be ignored.

These data elements are set during server initialization and are not changed thereafter. See section 3.2.3.

Note It is possible to implement the conceptual data defined in this section using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

Copyright © 2008 Microsoft Corporation.

3.2.1.2 Session State

An MSDTC Connection Manager: OleTx Transports Protocol partner MUST maintain a session table (a table of session objects) keyed by the contact identifier (CID) field of the Name field referenced by each session object. Each partner maintains a table of the sessions in progress. This table grows and shrinks as sessions are established and terminated. A session object MUST maintain the following data elements:

Name: A name object that contains contact information for the partner.

Version: A BOUND_VERSION_SET structure representing the session values negotiated between the two participants in the session.

Binding Handle: An RPC binding handle to the partner.

Context Handle: The RPC context handle associated with this session for the other partner.

Timers: Each session has two timers: a Session Setup timer and a Session Teardown timer.

State: The current state of the session. The state of the session MUST be one of the following values:

- Connecting
- Confirming Connection
- Active
- Requesting Teardown
- Teardown

The valid state transitions are described by one of the two following state diagrams, depending on whether the partner is the primary partner in the session or not. Only a secondary partner has the option to enter the Requesting Teardown state.

Copyright © 2008 Microsoft Corporation.

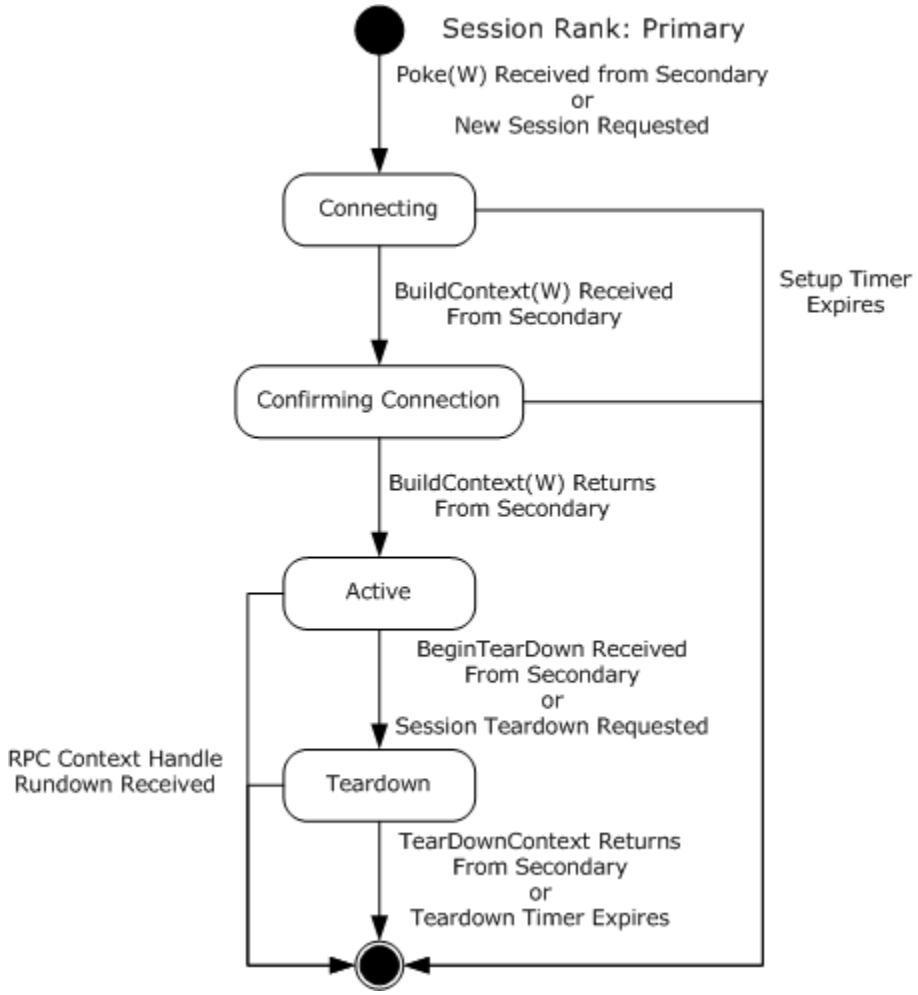


Figure 3: Primary session state

Copyright © 2008 Microsoft Corporation.

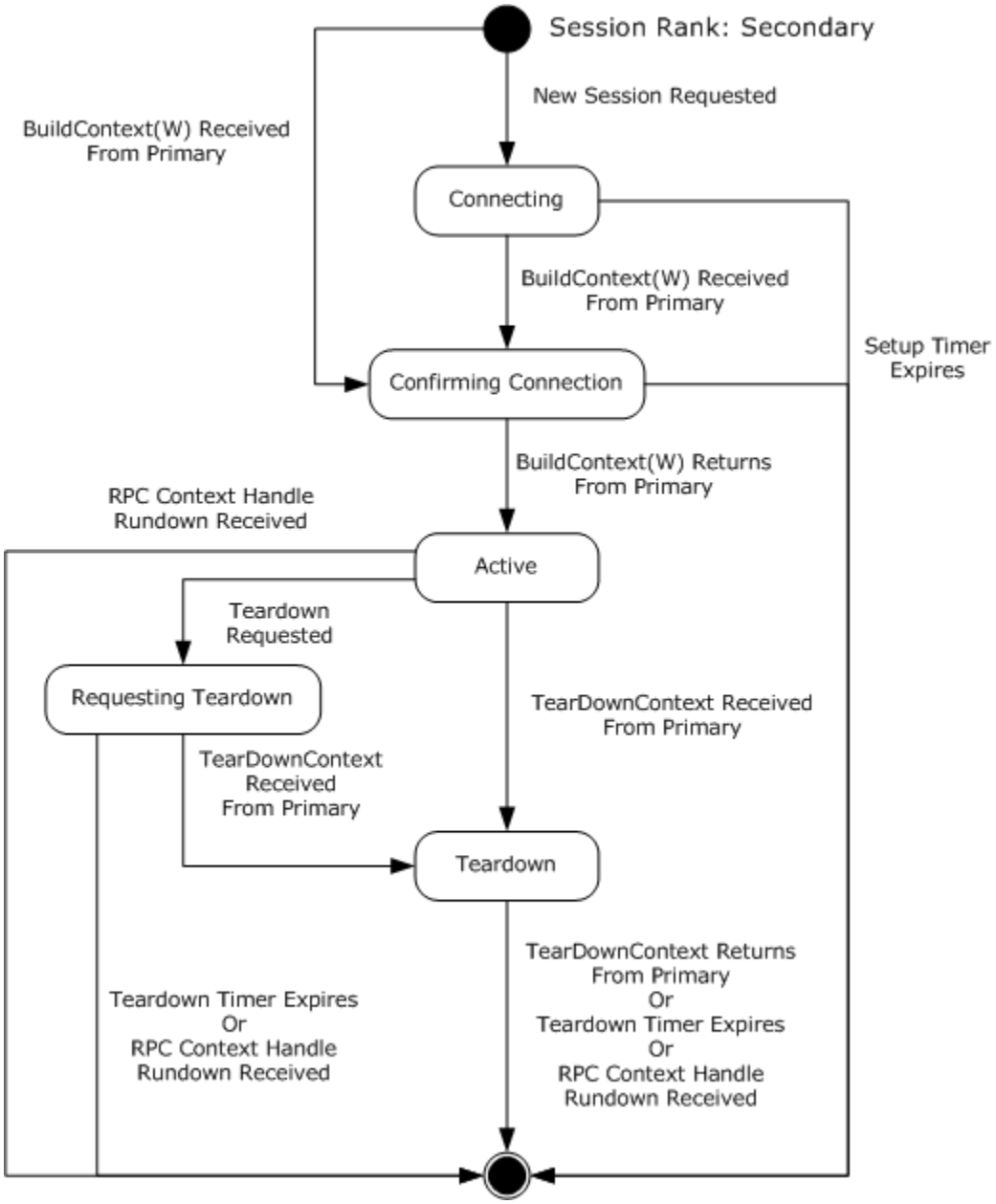


Figure 4: Secondary session state

Note It is possible to implement the conceptual data defined in this section using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

3.2.1.3 Cleaning Up a Session Object

When a session object is removed from the session table, it MUST be cleaned up as follows:

1. Any outstanding remote procedure call (RPC) associated with the session object MUST be canceled; this includes calls to **BuildContext**, **BuildContextW**, **Poke**, **PokeW**,

Copyright © 2008 Microsoft Corporation.

BeginTearDown, and **TearDownContext** that are being used to establish or tear down the session represented by the session object.

2. All active timers associated with the session object MUST be canceled.

3. The RPC binding handle stored in the session object MUST be released if it has been allocated. For RPC binding handles, see [C706].

4. The RPC context handle stored in the session object MUST be released if it has been allocated. For RPC context handles, see [C706].

3.2.1.4 Name Object

A name object MUST contain the following data elements:

Hostname: The NetBIOS name of the machine on which the partner is listening. For NetBIOS, see [NETBEUI], [RFC1001], and [RFC1002].

CID: The contact identifier (CID) of the remote partner.

Protocols: A list of the RPC network protocols supported by the partner.

Note It is possible to implement the conceptual data defined in this section using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

3.2.1.4.1 Name Object Comparison

Two name objects are considered equal if (and only if) their contact identifiers (CIDs) are identical **GUIDs**, and the Hostname fields are identical NetBIOS host names. For NetBIOS, see [NETBEUI] and [RFC1001].

3.2.2 Timers

An implementation of the MSDTC Connection Manager: OleTx Transports Protocol MUST provide Session Setup timers and Session Teardown timers. Each session object is associated with a pair of these timers.

3.2.2.1 Session Setup Timer

There is an instance of this timer corresponding to each session object. This timer MUST be set when the associated session enters the Connecting state or the Confirming Connection state, and is canceled when the session enters the Active state. The default value of the timer is specific to the implementation.<5>

3.2.2.2 Session Teardown Timer

There is an instance of this timer corresponding to each session object. This timer MUST be set when the associated session enters the Teardown state, and is canceled when the session leaves that state. The default value of the timer is specific to the implementation.<6>

3.2.3 Initialization

Each MSDTC Connection Manager: OleTx Transports Protocol partner is explicitly initialized with the following data elements, as described in 3.2.1.1:

- A Local Name object.

Copyright © 2008 Microsoft Corporation.

- The Minimum and Maximum Level 2 Version Numbers.
- The Minimum and Maximum Level 3 Version Numbers.
- A Security Level.

Initially, the session table, described in 3.2.1.2, is empty. At initialization, the partner MUST also perform the following actions:

- Set the Minimum and Maximum Level 1 Version Numbers as follows:
  - If the partner implements the MS-CMPO 1.1 protocol version, the Minimum Level 1 Version Number MUST be set to 0x00000001 and the Maximum Level 1 Version Number MUST be set to 0x00000002.
  - Otherwise, if the partner implements only the MS-CMPO 1.0 protocol version, both the Minimum and Maximum Level 1 Version Number MUST be set to 0x00000001.
- Create a dynamic endpoint on each of its supported RPC protocols, and register the interface with the RPC endpoint mapper. When it performs this registration, it specifies its local contact identifier (CID) as the object identifier. See [C706].
- Create an empty session table and assign it to the Session Table field.
- Obtain the client RPC security settings from an implementation specific source, and assign them to the Client Security Settings data field.
- Obtain the server RPC security settings from an implementation specific source, and assign them to the Server Security Settings data field.
- Set up the server RPC security using the Server Security Settings, and start the server to listen for RPC calls.<7>

3.2.4 Message Processing Events and Sequencing Rules

The MSDTC Connection Manager: OleTx Transports Protocol SHOULD indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 5.0, as specified in [MS-RPCE] section 3.<8>

This protocol MUST indicate to the RPC runtime via the strict_context_handle attribute that it is to reject use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Methods in RPC Opnum Order

Method	Description
Poke	Opnum: 0
BuildContext	Opnum: 1
NegotiateResources	Opnum: 2
SendReceive	Opnum: 3
TearDownContext	Opnum: 4
BeginTearDown	Opnum: 5

Copyright © 2008 Microsoft Corporation.

Method	Description
PokeW	Opnum: 6
BuildContextW	Opnum: 7

3.2.4.1 Poke (Opnum 0)

The **Poke** method is used by a secondary partner to request the primary partner session initiation. The parameter values specified in the call identify both participants.

```
HRESULT Poke(
  [in] handle_t hBinding,
  [in] SESSION_RANK sRank,
  [in, string, range(GUID_LENGTH, GUID_LENGTH)]
    unsigned char pszCalleeUuid,
  [in, string, range(1, MAX_COMPUTERNAME_LENGTH+1)]
    unsigned char pszHostName,
  [in, string, range(GUID_LENGTH, GUID_LENGTH)]
    unsigned char pszUuidString,
  [in, range(sizeof(BIND_INFO_BLOB),sizeof(BIND_INFO_BLOB))]
    DWORD dwcbSizeOfBlob,
  [in, size_is(dwcbSizeOfBlob)] unsigned char rguchBlob
);
```

hBinding: The RPC primitive binding handle of the partner receiving the call, as specified in [C706] part 3.

sRank: The session rank of the partner making the call. This parameter MUST be set to 0x02 (SRANK_SECONDARY).

Value	Meaning
SRANK_SECONDARY 0x02	The caller is the secondary participant.

pszCalleeUuid: The ANSI string form of the primary partner contact identifier (CID). The contact identifier (CID) MUST match the contact identifier (CID) in the primary partner local name object, and MUST be formatted as a string, as specified in [C706] Appendix A. The contact identifier (CID) is a globally unique identifier (GUID).

pszHostName: The ANSI string form of the caller's host name. This host name identifies the machine on which the caller's instance of the MSDTC Connection Manager: OleTx Transports Protocol is running. This value is used by the primary participant to establish the RPC binding handle for its subsequent call to **BuildContext**. This MUST be a NetBIOS name. For NetBIOS, see [NETBEUI], [RFC1001], and [RFC1002].

pszUuidString: The ANSI string form of the caller's contact identifier (CID). This contact identifier (CID) identifies the caller's instance of the MSDTC Connection Manager: OleTx Transports Protocol. It MUST match the contact identifier (CID) in the caller's local name object, and it MUST be formatted as a string, as specified in [C706] Appendix A. This value is used by the primary participant to establish the RPC binding handle for its subsequent call to **BuildContext**.

Copyright © 2008 Microsoft Corporation.

dwcbSizeOfBlob: The count, in bytes, of the size of the binding info structure. This parameter MUST be set to 8.

rguchBlob: A byte array containing a BIND_INFO_BLOB structure specifying the transport protocols supported. This information is used to build the RPC binding for the reverse connection.

Return Values: This method MUST return 0 on success. On failure, it MUST return an implementation-specific **HRESULT**. A client MUST NOT have behavior that is dependent on implementation-specific failure **HRESULT** values. The standard Windows errors are defined in [MS-ERREF] section 4.

Return value/code	Description
0x00000000 ERROR_STATUS	The return value indicates success.

The opnum field value for this method is 0.

Poke SHOULD NOT be invoked on a secondary partner. If it is, the secondary partner MUST respond by making a **Poke** callback on the primary partner. See section 3.3. The parameters to the **Poke** call MUST be calculated from the incoming parameters and the secondary partner's local name object; specifically, the pszCalleeUuid parameter MUST be set to the value of the pszUuidString parameter, the pszHostName parameter MUST be the Hostname field of the secondary partner's local name object, and the pszUuidString parameter MUST be the string form of the contact identifier (CID) field of the secondary partner's local name object. The secondary partner MAY return from the **Poke** method before this recursive call has completed.<9>

When **Poke** is invoked on a primary partner, the primary partner MUST construct a name object using the host name specified in the pszHostName parameter, the contact identifier (CID) specified in the pszUuidString parameter, and the RPC protocols specified in the grbitComProtocols field of the BindInfo field of the BIND_INFO_BLOB structure.

The primary partner MUST use this name object to check whether or not an existing session with a matching name object already exists in the session table. If an existing session is found, the primary partner MUST return an error code.<10> Otherwise, a new session object MUST be created and added to the session table. The new session object MUST be initialized with the created name object. An RPC binding handle to the secondary partner MUST be created and stored in the session object. For binding handles, see [C706]. The State field MUST be set to Connecting, and the Session Setup timer MUST be set; all other fields MUST be initialized to 0.

At this point, the primary partner MAY return success from the method provided it continues to perform the following actions; it does not have to wait until the entire process is completed.<11>

After initializing the session object and adding it to the session table, the primary partner MUST attempt to call either the **BuildContextW** method or the **BuildContext** method on the secondary partner with the RPC binding handle stored in the session object. For details on making calls to a partner, see section 3.3. If the secondary partner does not support the **BuildContextW** method, the primary partner MUST call the **BuildContext** method. If the secondary partner does support the **BuildContextW** method, the primary partner MUST NOT call the **BuildContext** method.

During this call, the secondary partner will make a nested, synchronous callback to the primary partner to complete the session establishment. See section 3.3.6.1.1.

If the call completes with an error code other than 0x80000172 (E_CM_VERSION_SET_NOTSUPPORTED), the partner SHOULD retry the call until the Session Setup

Copyright © 2008 Microsoft Corporation.

timer associated with the session expires. If the call completes with an error code of 0x80000172 (E_CM_VERSION_SET_NOTSUPPORTED), the partner SHOULD NOT retry the call.<12>

If the call completes successfully, the primary partner MUST examine the State field of the session object; if the value is "Confirming Connection", it MUST set the state of the session object to Active and cancel the Session Setup timer associated with that session object. If the Session Setup timer expires before the call completes successfully, if the call completes with error code 0x80000712, or if the State field of the session object is not "Confirming Connection", the primary partner MUST remove the session object from the session table and clean it up. For instructions on cleaning up a session object, see section 3.2.1.3.

3.2.4.2 BuildContext (Opnum 1)

The **BuildContext** method is invoked by either a primary partner or a secondary partner. When invoked by a primary partner, the **BuildContext** method requests that the secondary partner begin the next step of establishing a session. When invoked by a secondary partner, the **BuildContext** method requests that the primary partner complete the establishment of the session.

```
HRESULT BuildContext(
  [in] handle_t hBinding,
  [in] SESSION_RANK sRank,
  [in] BIND_VERSION_SET BindVersionSet,
  [in, string, range(GUID_LENGTH,GUID_LENGTH)]
    unsigned char pszCalleeUuid[ ],
  [in, string, range(1,MAX_COMPUTERNAME_LENGTH+1)]
    unsigned char pszHostName[ ],
  [in, string, range(GUID_LENGTH,GUID_LENGTH)]
    unsigned char pszUuidString[ ],
  [in, string, range(GUID_LENGTH,GUID_LENGTH)]
    unsigned char pszGuidIn[ ],
  [in, out, string, range(GUID_LENGTH,GUID_LENGTH)]
    unsigned char pszGuidOut[ ],
  [in, out] BOUND_VERSION_SET* pBoundVersionSet,
  [in, range(sizeof(BIND_INFO_BLOB), sizeof(BIND_INFO_BLOB))]
    DWORD dwcbSizeOfBlob,
  [in, size_is(dwcbSizeOfBlob)] unsigned char rguchBlob[ ],
  [out] PPCONTEXT_HANDLE ppHandle
);
```

hBinding: RPC primitive binding handle for the connection, as specified in [C706] part 3.

sRank: The session rank of the partner making the call. It MUST be one of the following values:

Value	Meaning
SRANK_PRIMARY 1	The caller is the primary partner in this session. The callee MUST be a secondary partner in this session, and the caller MUST be a primary partner in this session.
SRANK_SECONDARY 2	The caller is the secondary partner in this session. The callee MUST be a primary partner in this session, and the caller MUST be a secondary partner in this session.

BindVersionSet: A **BIND_VERSION_SET** structure that contains the minimum and maximum versions supported by the partner, as specified by the Minimum and Maximum Level 1 Version

Copyright © 2008 Microsoft Corporation.

Numbers, the Minimum and Maximum Level 2 Version Numbers, and the Minimum and Maximum Level 3 Version Numbers (see 3.2.1.1).

pszCalleeUuid: An ANSI string containing the callee's contact identifier (CID) in the form of a globally unique identifier (GUID). The contact identifier (CID) MUST match the contact identifier (CID) in the callee's local name object, and MUST be formatted into a string, as specified in [C706] Appendix A.

pszHostName: The ANSI string form of the caller's host name. This host name identifies the machine in which the caller's instance of the MSDTC Connection Manager: OleTx Transports Protocol is running. This MUST be a NetBIOS name. For NetBIOS, see [NETBEUI], [RFC1001], and [RFC1002].

If this is the primary partner call, this value is used by the called secondary partner to establish the RPC binding handle for its corresponding call to **BuildContext**.

pszUuidString: The ANSI string form of the caller's contact identifier (CID) in the form of a GUID. This contact identifier (CID) identifies the caller's instance of the MSDTC Connection Manager: OleTx Transports Protocol. It MUST match the contact identifier (CID) in the caller's local name object, and MUST be formatted into a string, as specified in [C706] Appendix A.

If this is the primary participant's call, this value is used by the called secondary participant to establish the RPC binding handle for its corresponding call to **BuildContext**.

pszGuidIn: A string form of a UUID that represents a unique identifier for this bind attempt. The UUID MUST be formatted as a string, as specified in [C706] Appendix A.

For the primary participant's call to **BuildContext**, this is a new GUID generated by the primary partner to uniquely identify the session. For the secondary partner's call back to the primary partner, this MUST be the parameter value from the primary partner's call to the secondary partner.

pszGuidOut: A string form of a UUID (as specified in [C706] Appendix A) that represents a unique identifier for this bind attempt. On input, it MUST be set to 00000000-0000-0000-0000-000000000000. On return, if the bind attempt is ultimately successful, it MUST be equal to the value of *pszGuidIn*. Otherwise, if the bind attempt is ultimately unsuccessful, it MUST be set to 00000000-0000-0000-0000-000000000000 on return.

pBoundVersionSet: A pointer to a **BOUND_VERSION_SET** structure. When the method is called, every field of the **BOUND_VERSION_SET** structure MUST be initialized to zero. On successful completion, the caller receives a **BOUND_VERSION_SET** on return.

dwcbSizeOfBlob: The count in bytes of the size of the binding info structure. This parameter MUST be set to the size of the BIND_INFO_BLOB, 8.

rguchBlob: A byte array containing the BIND_INFO_BLOB structure specifying the supported transport protocols. This information is used to build the RPC binding for the reverse connection.

ppHandle: On successful return, an RPC context handle that correlates with the session object created by (or referenced by) this method. For RPC context handles, see [C706].

Return Values: This method MUST return zero on success. On failure, it MUST return either 0x80000172 (E_CM_VERSION_SET_NOTSUPPORTED) or an implementation-specific **HRESULT**. A client SHOULD distinguish between 0x80000172 and other error codes, as specified in **BuildContext**, but MUST NOT have behavior that is dependent on implementation-specific failure HRESULT values.

Copyright © 2008 Microsoft Corporation.

The standard Windows errors are defined in section 4 of [MS-ERREF].

Return value/code	Description
0x00000000 ERROR_STATUS	The return value indicates success.
0x80000172 E_CM_VERSION_SET_NOTSUPPORTED	

The opnum field value for this method is 1. For more information, see [C706].

This method has different effects depending on the value of the sRank parameter.

For the structure and sequence of data on the wire, see [C706] Transfer Syntax Network Data Representation (NDR) topics.

3.2.4.2.1 Primary

If the *sRank* parameter is **SRANK_PRIMARY**, the caller MUST be a primary partner, and the callee MUST be a secondary partner. The primary partner is acting as the client for this session, as specified in section 3.3.6.1. The session object has already been created on the primary partner, and its state has been set to Connecting.

The secondary partner MUST construct a name object using the host name specified in the *pszHostName* parameter, the contact identifier (CID) specified in the *pszUuidString* parameter, and the RPC protocols specified in the grbitComProtocols field of the BindInfo field of the BIND_INFO_BLOB structure contained in the rguchBlob.

The secondary partner MUST use this name object to check whether an existing session with a matching name object already exists in the session table. If an existing session object is found (which would occur if the secondary partner initiated the connection through a call to the **Poke** method or the **PokeW** method), the secondary partner MUST check the State field of the session object. If the value is not Connecting, the secondary partner MUST return an error code.<13> If an existing session object is not found, a new session object MUST be created and added to the session table. Regardless of whether the session object was found or created, the State field of the session object MUST be set to Confirming Connection. If a new session object was created, it MUST be initialized with the name object.

Next, the secondary partner MUST calculate the *pBoundVersionSet* parameter as follows:

- The dwLevelOneAccepted field MUST be set to the largest value such that:
  - It is greater than or equal to the larger of the two values:
    - dwMinLevelOne field of the *BindVersionSet* parameter.
    - Local Minimum Level One Version Number.
  - It is less than or equal to the lesser of the two values:
    - dwMaxLevelOne field of the *BindVersionSet* parameter.
    - Maximum Level One Version Number.

Copyright © 2008 Microsoft Corporation.

If no such value exists, then the function MUST return with the 0x80000172 (E_CM_VERSION_SET_NOTSUPPORTED) error, and the cleanup steps described in the following list MUST be followed.

- The dwLevelTwoAccepted field MUST be set to the largest value such that:
  - It is greater than or equal to the larger of the two values:
    - dwMinLevelTwo field of the *BindVersionSet* parameter.
    - Local Minimum Level Two Version Number.
  - It is less than or equal to the lesser of the two values:
    - dwMaxLevelTwo field of the *BindVersionSet* parameter.
    - Maximum Level Two Version Number.

  If no such value exists, then the function MUST return with the 0x80000172 (E_CM_VERSION_SET_NOTSUPPORTED) error, and the following cleanup steps MUST be followed.
- The dwLevelThreeAccepted field MUST be set to the largest value such that:
  - It is greater than or equal to the larger of the two values:
    - dwMinLevelThree field of the *BindVersionSet* parameter.
    - Local Minimum Level Three Version Number.
  - It is less than or equal to the lesser of the two values:
    - dwMaxLevelThree field of the *BindVersionSet* parameter.
    - Maximum Level Three Version Number.

  If no such value exists, then the function MUST return with the 0x80000172 (E_CM_VERSION_SET_NOTSUPPORTED) error, and the following cleanup steps MUST be followed.

The *pBoundVersionSet* parameter calculated above contains the maximum protocol versions supported by both partners for the MSDTC Connection Manager: OleTx Transports Protocol implementation, and the level-two and level-three protocol implementations layered on top of that implementation (see also 3.2.1.1). These represent the negotiated protocol versions that MUST be used in the respective protocol communications.

If any of the previously described operations fail, the secondary partner MUST remove the session object from the session table and clean it up. See section 3.2.1.3. After cleaning up the session object, the secondary partner MUST return from this method with the specified error code.

If the previously described calculations succeed, a copy of the **BOUND_VERSION_SET** structure MUST also be stored in the Version field of the session object. Once this is done, the secondary partner MUST start the Session Setup timer associated with that session object if it has not already been started. The Session Setup timer will not have been started if the session establishment began with the primary partner. In this case, this method call is the first time the secondary partner has considered this session.

Copyright © 2008 Microsoft Corporation.

An RPC binding handle to the primary partner MUST be created and stored in the session object. For binding handles, see [C706]. The secondary partner MUST attempt to call either the **BuildContextW** method or the **BuildContext** method on the primary partner using the binding handle stored in the session object. For making calls to a partner, see section 3.3. If the primary partner does not support the **BuildContextW** method, the secondary partner MUST call the **BuildContext** method. If the primary partner does support the **BuildContextW** method, the secondary partner MUST NOT call the **BuildContext** method. The secondary partner MUST NOT return from the current call to **BuildContext** until the nested call to **BuildContext** or **BuildContextW** has completed.

If the incoming RPC is authenticated, the secondary partner MAY use the authenticated identity of the caller as the server principal name for performing mutual authentication on the nested call. For performing the call, see section 3.3. For security, see section 2.1.3.

If the nested call completes successfully, the secondary partner MUST set the state of the session object to Active, store the received context handle in the associated session object, and cancel the Session Setup timer associated with that session object. It MUST set the *contextHandle* parameter to a context handle (see [C706]) that identifies the session object, and then return from the method with the S_OK code.

If the nested call completes unsuccessfully with any error code other than 0x80000172 (E_CM_VERSION_SET_NOTSUPPORTED), the secondary partner SHOULD retry the nested call until the Session Setup timer associated with the session expires.<14>

If the nested call completes with the error code 0x80000172, or the Session Setup timer expires, the secondary partner MUST remove the session object from the session table and clean it up. See section 3.2.1.3. After cleaning up the session object, the secondary partner MUST return an error code from the method.

3.2.4.2.2 Secondary

If the *sRank* parameter is **SRANK_SECONDARY**, the caller MUST be a secondary partner, and the callee MUST be a primary partner. The primary partner MUST construct a name object using the host name specified in the pszHostName parameter, the contact identifier (CID) specified in the pszUuidString parameter, and the RPC protocols specified in the grbitComProtocols field of the BindInfo field of the BIND_INFO_BLOB structure contained in the rguchBlob.

The primary partner MUST use this name object to check whether or not an existing session with a matching name object already exists in the session table, and whether or not the State field of that session object is set to Connecting. If an existing session cannot be found, or if the state of the session object is not Connecting, the primary partner MUST return an error code.<15>

Next, the primary partner MUST compute the *pBoundVersionSet* parameter, as specified in section 3.2.4.2.1. If the computation fails, the session object MUST be cleaned up, as specified in section 3.2.4.2.1. This value MUST also be stored in the Version field of the session object. Finally, the primary partner MUST set the State field of the session object to Confirming Connection, and then return from the method with the S_OK code.

3.2.4.3 NegotiateResources (Opnum 2)

The **NegotiateResources** method is invoked by one partner to request that the other partner allocate resources for future use.

```
HRESULT NegotiateResources(
  [in] PCONTEXT_HANDLE phContext,
```


Copyright © 2008 Microsoft Corporation.

```
  [in] DWORD resourceType,
  [in] DWORD dwcRequested,
  [in, out] DWORD* pdwcAccepted
);
```

phContext: An RPC context handle that is correlated with a session object that is in the Active state. For context handles, see [C706].

resourceType: An unsigned 32-bit integer identifying the resource type to be negotiated.

dwcRequested: An unsigned 32-bit integer that specifies the number of resources to allocate. This value MUST be greater than 0x00 and less than 1,000.

pdwcAccepted: A pointer to an unsigned 32-bit integer that receives the number of resources that were allocated on behalf of the caller. This value MAY be smaller than the value of dwcRequested if the partner was incapable of allocating all of the requested resources.<16> On input, this value MUST be set to 0x00.

Return Values: This method MUST return zero (0x00000000) on success. On failure, it MUST return an implementation-specific HRESULT.

Note A client MUST NOT exhibit behavior observable on the wire that is dependent on implementation-specific failure HRESULT values.

The opnum field value for this method is 2. See [C706].

For the structure and sequence of data on the wire, see [C706] Transfer Syntax Network Data Representation (NDR) topics.

On receiving this method call, the receiving partner MUST verify that the contextHandle parameter is a valid context handle associated with a session object that is in the Active state. For context handles, see [C706]. If the context handle is invalid, or if the session object is not in the Active state, the partner MUST return from this method with an error code.<17>

Otherwise, the operation of this method is determined by the protocol layered on top of the MSDTC Connection Manager: OleTx Transports Protocol; it is this upper-layer protocol that defines the range of valid values for the resourceType parameter. If the resourceType parameter does not identify a valid resource, the partner MUST return an error from this method.<18> If the upper-layer protocol cannot reserve any resources at all, the partner MUST return 0x80000127 (E_CM_OUTOFRESOURCES). Otherwise, if at least one resource is allocated, the partner MUST set the pdwcAccepted parameter to the number of resources allocated by this request, and then return S_OK.

3.2.4.4 SendReceive (Opnum 3)

The **SendReceive** method is invoked by one partner to transmit messages to the other partner. Both the primary and the secondary participants have the option to call this method multiple times after a session has been established between them.

```
HRESULT SendReceive(
  [in] PCONTEXT_HANDLE phContext,
  [in, range(1, 4095)] DWORD dwcMessages,
  [in, range(32, 0x14000)] DWORD dwcbSizeOfBoxCar,
  [in, size_is(dwcbSizeOfBoxCar)]
    unsigned char rguchBoxCar
```


Copyright © 2008 Microsoft Corporation.

```
);
```

phContext: An RPC context handle, returned by a call to **BuildContext** or **BuildContextW**, correlated with a session object in the Active state. For context handles, see [C706].

dwcMessages: An unsigned 32-bit integer specifying the number of messages being sent.

dwcbSizeOfBoxCar: Size in bytes of the box car specified by *rguchBoxCar*.

rguchBoxCar: An array of bytes that contains the messages being sent.

Return Values: This method MUST return 0 on success. On failure, it MUST return an implementation-specific **HRESULT**. A client MUST NOT have behavior that is dependent on implementation-specific failure **HRESULT** values. The standard Windows errors are defined in [MS-ERREF] section 4.

Return value/code	Description
0x00000000 ERROR_STATUS	The return value indicates success.

The opnum field value for this method is 3, as specified in [C706].

For the structure and sequence of data on the wire, see [C706] section [C706-Ch14TransSyntaxNDR].

On receiving this method call, the receiving partner MUST verify that the contextHandle parameter is a valid context handle associated with a session object that is in the Active state. For context handles, see [C706]. If the context handle is invalid, or if the session object is not in the Active state, the partner MUST return from this method with an error code.<19>

Otherwise, the operation of this method is determined by the protocol layered on top of the MSDTC Connection Manager: OleTx Transports Protocol; the session object, the count of messages, and the byte array MUST be presented to the higher-layer protocol. It is this higher-layer protocol that defines the format of the *rguchBoxCar* buffer and the messages contained therein. Similarly, any correlation between the *dwcMessages* parameter and the contents of the *rguchBoxCar* buffer lies strictly in the domain of the upper-layer protocol.

3.2.4.5 TearDownContext (Opnum 4)

The **TearDownContext** method is invoked by either a primary partner or a secondary partner. When invoked by a primary partner, the **TearDownContext** method requests that the secondary partner begin the next step of tearing down a session. When invoked by a secondary partner, the **TearDownContext** method requests that the primary partner complete the teardown of the session. The **MIDL** syntax of the method is as follows.

```
HRESULT TearDownContext(
  [in, out] PPCONTEXT_HANDLE contextHandle,
```


Copyright © 2008 Microsoft Corporation.

```
  [in] SESSION_RANK sRank,
  [in] TEARDOWN_TYPE tearDownType
);
```

contextHandle: An RPC context handle that is correlated with a session object that is in the Active state. For context handles, see [C706].

sRank: A **SESSION_RANK** enumerated value indicating whether the teardown request is being made by a primary partner or secondary partner. The only valid teardown request should come from a primary host.

Value	Meaning
SRANK_PRIMARY 0x01	The caller is the primary partner in this session. The callee MUST be a secondary partner in this session, and the caller MUST be a primary partner in this session.
SRANK_SECONDARY 0x02	The caller is the secondary partner in this session. The callee MUST be a primary partner in this session, and the caller MUST be a secondary partner in this session.

tearDownType: The reason for tearing down the session. It MUST be one of the following values.

Value	Meaning
TT_FORCE 0x00	The session is being forcefully torn down.
TT_PROBLEM 0x02	The session is being torn down because an error has occurred.

Return Values: This method MUST return zero (0x00000000) on success. On failure, it MUST return an implementation-specific HRESULT.

Note A client MUST NOT exhibit behavior observable on the wire that is dependent on implementation-specific failure HRESULT values.

On receiving this method call, the receiving partner MUST verify that the contextHandle parameter is a valid context handle associated with a session object that is in the Active, Requesting Teardown, or Teardown state. For context handles, see [C706]. If the context handle is invalid, or if the session object is not in the Active state, the partner MUST return from this method with an error code.<20>

Thereafter, the method has a different effect depending on the value of the *sRank* parameter and the value of the teardownType parameter.

3.2.4.5.1 Problem

If the teardownType parameter is TT_PROBLEM, the receiving partner MUST invalidate the context handle, remove the associated session object from the session table, and close the binding handle associated with the session object. (See [C706].) Once this has been done, the upper-layer protocol MUST be notified that a problem teardown has occurred, and provide the upper-layer protocol with the session object.

Copyright © 2008 Microsoft Corporation.

3.2.4.5.2 Primary

If the *teardownType* parameter is not TT_PROBLEM, and the **sRank** parameter is SRANK_PRIMARY, the caller MUST be a primary partner, and the callee MUST be a secondary partner.

The secondary partner MUST set the state of the session object associated with the context handle to Teardown. After setting the state, it MUST free the context handle associated with the session and return S_OK from the method.

In addition, it MUST start the Session Teardown timer associated with that session object if it has not already been started, and attempt to call the TearDown method on the primary partner. When the call completes, regardless of whether it was successful or not, or when the Session Teardown timer expires, the secondary partner MUST close the binding handle of the session object, cancel the Session Teardown timer, and remove the session object from the session table. (See [C706].) Once this has been done, the upper-layer protocol MUST be notified that a forced teardown has occurred, and provide the upper-layer protocol with the session object.

The secondary partner MAY choose to perform these actions asynchronously.<21>

3.2.4.5.3 Secondary

If the teardownType parameter is not TT_PROBLEM, and the **sRank** parameter is SRANK_SECONDARY, the caller MUST be a secondary partner, and the callee MUST be a primary partner.

The primary partner MUST close the binding handle of the session object, cancel any active timers associated with the session object, and remove the session object from the session table. The primary partner MUST then free the context handle associated with that session and return S_OK from the method. (See [C706].) Once this has been done, the upper-layer protocol MUST be notified that a forced teardown has occurred, and provide the upper-layer protocol with the session object.

3.2.4.6 BeginTearDown (Opnum 5)

The **BeginTearDown** method is invoked by a secondary partner to request that a primary partner begin session teardown.

```
HRESULT BeginTearDown(
  [in] PCONTEXT_HANDLE contextHandle,
  [in] TEARDOWN_TYPE tearDownType
);
```

contextHandle: An RPC context handle that is correlated with a session object that is in the Active state. For context handles, see [C706].

tearDownType: The reason for tearing down the session. It MUST be set to 0x00 (TT_FORCE).

Value	Meaning
0x00	TT_FORCE

Return Values: This method MUST return zero (0x00000000) on success. On failure, it MUST return an implementation-specific HRESULT.

Copyright © 2008 Microsoft Corporation.

Note A client MUST NOT exhibit behavior observable on the wire that is dependent on implementation-specific failure HRESULT values.

BeginTearDown MUST NOT be invoked on a secondary partner.

When this method is invoked on a primary partner, the receiving partner MUST verify that the contextHandle parameter is a valid context handle associated with a session object that is in the Active or Teardown state. For context handles, see [C706]. If the context handle is invalid, or if the session object is not in the Active state, the partner MUST return from this method with an error code.<22>

If the session object is in the Teardown state, the primary partner MUST immediately return from the method with S_OK. Otherwise, the primary partner MUST set the state of the session object associated with the context handle to Teardown and return S_OK from the method. Also, it MUST start the Session Teardown timer associated with that session object and attempt to call the **TearDownContext** method on the secondary partner. The secondary partner MAY choose to perform these actions asynchronously.<23>

3.2.4.7 PokeW (Opnum 6)

The **PokeW** method is equivalent in all ways to the **Poke** method except that its string parameters are encoded in UTF-16.

```
HRESULT PokeW(
  [in] handle_t hBinding,
  [in] SESSION_RANK sRank,
  [in, string, range(GUID_LENGTH, GUID_LENGTH)]
    wchar_t pwszCalleeUuid[ ],
  [in, string, range(1, MAX_COMPUTERNAME_LENGTH+1)]
    wchar_t pwszHostName[ ],
  [in, string, range(GUID_LENGTH, GUID_LENGTH)]
    wchar_t pwszUuidString[ ],
  [in, range(sizeof(BIND_INFO_BLOB),sizeof(BIND_INFO_BLOB))]
    DWORD dwcbSizeOfBlob,
  [in, size_is(dwcbSizeOfBlob)] unsigned char rguchBlob[ ]
);
```

hBinding: The RPC primitive binding handle, as specified in [C706] part 3.

sRank: The **SESSION_RANK** of the partner making the call. This parameter MUST be set to 0x02 (SRANK_SECONDARY).

Value	Meaning
SRANK_SECONDARY 0x02	The caller is the secondary participant.

pwszCalleeUuid: The string form of the primary partner contact identifier (CID). The contact identifier (CID) MUST match the contact identifier (CID) in the primary partner local name object, and MUST be formatted into a string, as specified in [C706] Appendix A.

pwszHostName: The string form of the caller's host name. This host name identifies the machine in which the caller's instance of the MSDTC Connection Manager: OleTx Transports Protocol is running. This MUST be a NetBIOS name. For NetBIOS, see [NETBEUI], [RFC1001], and [RFC1002].

Copyright © 2008 Microsoft Corporation.

pwszUuidString: The string form of the caller's contact identifier (CID). This contact identifier (CID) identifies the caller's instance of the MSDTC Connection Manager: OleTx Transports Protocol; it MUST match the contact identifier (CID) in the caller's local name object, and MUST be formatted into a string, as specified in [C706] Appendix A.

dwcbSizeOfBlob: The count, in bytes, of the size of the binding info structure. This parameter MUST be set to the size of the BIND_INFO_BLOB, 8.

rguchBlob: A byte array that contains a BIND_INFO_BLOB structure.

Return Values: This method MUST return zero (0x00000000) on success. On failure, it MUST return an implementation-specific HRESULT.

Note A client MUST NOT exhibit behavior observable on the wire that is dependent on implementation-specific failure HRESULT values.

Exceptions Thrown

No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

3.2.4.8 BuildContextW (Opnum 7)

The **BuildContextW** method is equivalent in all ways to the **BuildContext** method except that its string parameters are encoded in UTF-16. The Microsoft Interface Definition Language (MIDL) syntax of the method is as follows.

```
HRESULT BuildContextW(
  [in] handle_t hBinding,
  [in] SESSION_RANK sRank,
  [in] BIND_VERSION_SET BindVersionSet,
  [in, string, range(GUID_LENGTH, GUID_LENGTH)]
    wchar_t pwszCalleeUuid,
  [in, string, range(1, MAX_COMPUTERNAME_LENGTH+1)]
    wchar_t pwszHostName,
  [in, string, range(GUID_LENGTH, GUID_LENGTH)]
    wchar_t pwszUuidString,
  [in, string, range(GUID_LENGTH, GUID_LENGTH)]
    wchar_t pwszGuidIn,
  [in, out, string, range(GUID_LENGTH, GUID_LENGTH)]
    wchar_t pwszGuidOut,
  [in, out] BOUND_VERSION_SET* pBoundVersionSet,
  [in, range(sizeof(BIND_INFO_BLOB), sizeof(BIND_INFO_BLOB))]
    DWORD dwcbSizeOfBlob,
  [in, size_is(dwcbSizeOfBlob)] unsigned char rguchBlob,
  [out] PPCONTEXT_HANDLE ppHandle
);
```

hBinding: RPC primitive binding handle, as specified in [C706] part 3.

sRank: The rank of the caller.

Value	Meaning
SRANK_PRIMARY 0x01	The caller is the primary partner in this session. The callee MUST be a secondary partner in this session, and the caller MUST be a primary

Copyright © 2008 Microsoft Corporation.

Value	Meaning
	partner in this session.
SRANK_SECONDARY 0x02	The caller is the secondary partner in this session. The callee MUST be a primary partner in this session, and the caller MUST be a secondary partner in this session.

BindVersionSet: A **BIND_VERSION_SET** structure that contains the minimum and maximum versions supported by the partner.

pwszCalleeUuid: The string form of the callee's contact identifier (CID). The contact identifier (CID) MUST match the contact identifier (CID) in the callee's local name object, and MUST be formatted into a string, as specified in [C706] Appendix A.

pwszHostName: The string form of the caller's host name. This host name identifies the machine in which the caller's instance of the MSDTC Connection Manager: OleTx Transports Protocol is running. This MUST be a NetBIOS name. For NetBIOS, see [NETBEUI], [RFC1001], and [RFC1002].

pwszUuidString: The string form of the caller's contact identifier (CID). This contact identifier (CID) identifies the caller's instance of the MSDTC Connection Manager: OleTx Transports Protocol. This MUST match the contact identifier (CID) in the caller's local name object, and MUST be formatted into a string, as specified in [C706] Appendix A.

pwszGuidIn: A string form of a UUID that represents a unique identifier for this bind attempt. The UUID MUST be formatted into a string, as specified in [C706] Appendix A.

pwszGuidOut: A string form of a UUID (as specified in [C706] Appendix A) that represents a unique identifier for this bind attempt. On input, it MUST be set to 00000000-0000-0000-0000-000000000000. On return, if the bind attempt is ultimately successful, it MUST be equal to the value of pszGuidIn. Otherwise, if the bind attempt is ultimately unsuccessful, it MUST be set to 00000000-0000-0000-0000-000000000000 on return.

pBoundVersionSet: A pointer that points to a **BOUND_VERSION_SET** structure. When the method is called, every field of the **BOUND_VERSION_SET** structure MUST be initialized to zero. On successful completion, and the receives a **BOUND_VERSION_SET** on return.

dwcbSizeOfBlob: The count in bytes of the size of the binding info structure. This parameter MUST be set to the size of BIND_INFO_BLOB, 8.

rguchBlob: A byte array that contains a BIND_INFO_BLOB structure.

ppHandle: On successful return, an RPC context handle (see [C706]) that correlates with the session object created by, or referenced by, this method. See below.

Return Values: This method MUST return zero on success. On failure, it MUST return either 0x80000172 (E_CM_VERSION_SET_NOTSUPPORTED) or an implementation-specific **HRESULT**.

Note A client SHOULD distinguish between E_CM_VERSION_SET_NOTSUPPORTED and other error codes, as specified in **BuildContext**, but MUST NOT have behavior that is dependent on implementation-specific failure **HRESULT** values.

Exceptions Thrown

Copyright © 2008 Microsoft Corporation.

No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

3.2.5 Timer Events

Note that the events that follow are described as asynchronous with respect to the normal operation of the MSDTC Connection Manager: OleTx Transports Protocol. If events are implemented this way, it is the responsibility of the implementation to ensure that its state remains consistent.

3.2.5.1 Session Setup Timer

When the Session Setup timer expires, the partner SHOULD:<24>

- Cancel any outstanding call to **BuildContext** or **BuildContextW**.

When the Session Setup timer expires, the partner MUST:

1. Remove the associated session object from the session table, and close any context handle or binding handle stored in the session object. (See [C706].)
2. Return an error result from the current incoming call to **BuildContext** or **BuildContextW** from the partner identified by the name object associated with the timer's session object, if any.<25>
3. Return an error result to any upper-level protocol that is requesting a new session to the partner identified by the name object stored in the timer's corresponding session object, if any.

3.2.5.2 Session Teardown Timer

When the Session Teardown timer expires, the partner SHOULD:<26>

- Cancel any outstanding call to **TearDownContext**.

When the Session Teardown timer expires, the partner MUST:

1. Remove the associated session object from the session table, and close any context handle or binding handle stored in the session object. (See [C706].)
2. Return an error result from the current incoming call to **TearDownContext** from the partner identified by the name object associated with the timer's session object, if any.<27>
3. Report success to any upper-level protocol that is requesting a new session to the partner identified by the name object stored in the timer's session corresponding object, if any.

3.2.6 Other Local Events

3.2.6.1 Context Handle Rundown

When the RPC runtime indicates that a context handle associated with a session is being run down, the participant MUST remove the associated session object from the session table, and close any context handle or binding handle stored in the session object. (See [C706].) Once this has been done, the upper-layer protocol MUST be notified that a teardown has occurred, and provide the upper-layer protocol with the session object.

Note Context handle rundown MAY be asynchronous with respect to the normal operation of the protocol. It is the responsibility of the implementation to ensure that its state remains consistent.<28>

Copyright © 2008 Microsoft Corporation.

3.3 IXnRemote Client Details

3.3.1 Abstract Data Model

The abstract data model for an IXnRemote client MUST be identical to the abstract data model described in section 3.2.1.

3.3.2 Timers

The timers for an IXnRemote client MUST be identical to the timers described in section 3.2.2.

3.3.3 Initialization

Every IXnRemote client is also an IXnRemote server, and every IXnRemote server is also an IXnRemote client. The client initialization steps described here occur each time one partner contacts another; an MSDTC Connection Manager: OleTx Transports Protocol implementation takes the actions listed below once for each session object.

A client begins with a session object that contains a name object, which contains contact information for a remote partner. The name object is used to create an RPC binding handle (see [C706]) to the remote partner's RPC endpoint. The name object contributes to the binding as follows:

1. The binding's protocol sequence is taken from one of the entries in the Protocols list in the name object. The protocol MUST be one of the protocols supported by both partners, and is selected as specified in section 2.

2. The binding's host network address is specified as the Hostname in the name object.

3. The binding's object UUID is specified as the contact identifier (CID) in the name object.

This partial binding is resolved into a full binding using the RPC Endpoint Mapper service at the host network address. The same binding handle MUST be used for every call to the remote partner.

With regard to RPC security (see 3.2.1.1), a client communicating with a server MUST do the following:

- If the Security Level is Mutual Authentication (see [Link - section 3.2.1.1]), the client MUST create and use an authenticated RPC binding handle using the Security Provider security settings specified by the Client Security Settings data field.

- Otherwise, if the Security Level is Incoming Authentication or Fallback to Unsecured if Necessary, the client MUST first attempt to create and use an authenticated RPC binding handle using the Security Provider security settings specified by the Client Security Settings data field. If the RPC authentication establishment fails, the client MUST create and use an unsecured binding handle to communicate with the server.

- Otherwise, the Security Level MUST be Unsecured, and the client MUST create and use an unsecured binding handle to communicate with the server.

3.3.4 Message Processing Events and Sequencing Rules

This protocol SHOULD indicate to the RPC runtime that it is to perform a strict Network Data Representation (NDR) data consistency check at target level 5.0, as specified in [MS-RPCE] section 3.<29>

Copyright © 2008 Microsoft Corporation.

3.3.5 Timer Events

The handling for the timer events is identical to the behavior described in section 3.2.5.

3.3.6 Other Local Events

3.3.6.1 New Session Requested

When the upper-level protocol requests a new session, it provides the necessary information to the local partner to initialize a client instance of the MSDTC Connection Manager: OleTx Transports Protocol. The local partner then determines the session rank for the new session.

3.3.6.1.1 Primary

If the local partner is the primary partner, it MUST use the provided name object to check whether or not an existing session with a matching name object already exists in the session table. If an existing session is found, the primary partner MUST return an error code.<30> Otherwise, a new session object MUST be created and added to the session table.

The local partner MUST set the state of the session object to Connecting, and start the Session Setup timer associated with that session object. An RPC binding handle to the secondary partner MUST be created and stored in the session object (for binding handles, see [C706]). The local partner MUST attempt to call either the **BuildContextW** or **BuildContext** method on the secondary partner using the binding handle stored in the session object. (For making calls to a partner, see section 3.3.) The binding handle used to make the call MUST be stored in the session object. (For binding handles, see [C706].) If the secondary partner does not support the **BuildContextW** method, the primary partner MUST call the **BuildContext** method. If the secondary partner does support the **BuildContextW** method, the primary partner MUST NOT call the **BuildContext** method.

When the call completes with the 0x80000172 (E_CM_VERSION_SET_NOTSUPPORTED) error, the partner MUST report an error to the upper-level protocol. Otherwise, the partner SHOULD retry the call until the Session Setup timer associated with the session object expires.<31>

If an error is reported to the upper-layer protocol, the session object MUST be removed from the session table and cleaned up. For how to clean up a session object, see section 3.2.1.3.

3.3.6.1.2 Secondary

If the local partner is the secondary partner, it MUST use the provided name object to check whether or not an existing session with a matching name object already exists in the session table. If an existing session is found, the primary partner MUST return an error code.<32> Otherwise, a new session object MUST be created and added to the session table.

If the local partner is the secondary partner, it MUST start the Session Setup timer associated with the provided name object, and then make a call to either the **Poke** method or the **PokeW** method on the primary partner. (For making calls to a partner, see section 3.3.) If the primary partner does not support the **PokeW** method, the secondary partner MUST call the **Poke** method. If the primary partner does support the **PokeW** method, the secondary partner MUST NOT call the **Poke** method.

When the call completes with an error, the secondary partner SHOULD retry the call until the Session Setup timer associated with the provided name object expires.<33>

When the call completes successfully, the secondary partner MUST wait until a session object associated with the provided name object is in the session table and the state of that session object

Copyright © 2008 Microsoft Corporation.

is Active, or until the Session Setup timer associated with the provided name object expires. If the Session Setup timer expires, the session object MUST be removed from the session table and cleaned up. For how to clean up a session object, see section 3.2.1.3.

3.3.6.2 Forced Session Teardown Requested

When the upper-level protocol requests a forced session teardown, it indicates what session object it wants to issue the teardown on. The session object MUST be in the Active state.

If the local partner is the primary partner, it MUST set the State field of the session object to Teardown, and then issue a **TearDownContext** call on the secondary partner, specifying the contextHandle parameter to be the context handle from the session object, the teardownType parameter as 0x00 (TT_FORCE), and the sRank parameter as SRANK_PRIMARY.

If the local partner is the secondary partner, it MUST set the State field of the session object to Requesting Teardown, and then issue a **BeginTearDown** call on the primary partner. It MUST specify the contextHandle parameter to be the context handle from the session object, and the teardownType parameter as 0x00 (TT_FORCE).

Any error that occurs while processing this request MUST be ignored.

3.3.6.3 Problem Session Teardown Requested

When the upper-level protocol requests a problem session teardown, it indicates what session object it wants to issue the teardown on.

The local partner MUST start the Session Setup timer associated with the session, set the State field of the session object to Teardown, and issue a **TearDownContext** call on the remote partner, specifying the contextHandle parameter to be the context handle from the session object, the teardownType parameter as 0x02 (TT_PROBLEM), and the sRank parameter as either 0x01 (SRANK_PRIMARY) if the local partner is the primary partner, or 0x02 (SRANK_SECONDARY) if the local partner is the secondary partner.

When the call completes, regardless of whether it was successful or not, or when the Session Teardown timer expires, the local partner MUST remove the session object from the session table and clean up the session object. For how to clean up a session object, see section 3.2.1.3.

Any error that occurs while processing this request MUST be ignored.

3.3.6.4 Resource Allocation Requested

When the upper-level protocol requests resource allocation, it indicates what session object it wants to allocate resources from. It also provides an integer specifying the type of resource to be allocated, and the number of resources that it wants to allocate. The local partner MUST issue a **NegotiateResources** call on the remote partner, specifying the contextHandle parameter as the context handle from the session object, the resourceType parameter as the provided resource type, and the dwcRequested parameter as the number of resources being requested. If the request succeeds, the value of the pdwcAccepted parameter MUST be provided back to the upper-level protocol.

Any error that occurs while processing this request MUST be reported to the upper-level protocol.

3.3.6.5 Message Send Requested

When the upper-level protocol requests a message send, it indicates what session object it wants to send the messages on. It also provides an integer count of messages (between 1 and 4,095

Copyright © 2008 Microsoft Corporation.

inclusive) and the message data contained in a byte array (containing from 32 to 81,920 bytes). The local partner MUST issue a **SendReceive** call on the remote partner, specifying the contextHandle parameter as the context handle from the session object, the dwcMessages parameter as the count of messages, the dwcbSizeOfBoxCar parameter as the size of the message data byte array, and the rguchBoxCar parameter as the message data byte array.

Any error that occurs while processing this request MUST be reported to the upper-level protocol.

Copyright © 2008 Microsoft Corporation.

4 Transports Examples

To participate in an MSDTC Connection Manager: OleTx Transports Protocol session, a partner exposes an endpoint to its implementation of the IXnRemote interface. Each partner's endpoint is identified by its name object, which includes its NetBIOS machine name, supported RPC network protocols, and contact identifier (CID), as specified in section 3.2.1.4. To begin a session, the first partner needs to have knowledge of the second partner's name object.

From the second partner's contact identifier (CID), the first partner determines if it is the primary partner or secondary partner by performing a case-insensitive string comparison of the first partner's and second partner's contact identifiers (CIDs), as specified in [C706]. If the first partner's contact identifier (CID) string is greater than the second partner's contact identifier (CID) string, the first partner is the primary partner. If the first partner's contact identifier (CID) string is less than the second partner's contact identifier (CID) string, the first partner is the secondary partner.

A session is initiated by the primary partner sending a **BuildContext** (or **BuildContextW**) call to the secondary partner with sRank set to SRANK_PRIMARY. In response, the secondary partner sends a **BuildContext** call to the secondary partner with sRank set to SRANK_SECONDARY. When the primary partner accepts the **BuildContext** call from the secondary partner, the secondary partner returns success to the primary partner's **BuildContext** call. Because the first **BuildContext** call in the protocol handshake originates from the primary partner, the secondary partner is required to begin a session with the primary partner by calling **Poke** (or **PokeW**), which instructs the primary partner to send a **BuildContext** call to the secondary partner.

4.1 Initiating a Session as Primary Partner

In this example, the first partner is on Machine_1 with contact identifier (CID) b51996ef-c434-4f79-a288-56efd302fc8e, and the second partner is on Machine_2 with contact identifier (CID) a3afb37b-f64a-4e6c-9017-f6a96ba6f166. Therefore, the first partner assumes the role of the primary partner, and the second partner assumes the role of the secondary partner.

In this example, both partners support Unicode method calls. This example assumes that the primary partner does not have an existing session with the secondary partner, because only one session is allowed between any two partners.

Because this is a new session, the primary partner will create a new object with a newly generated session globally unique identifier (GUID). The session object is keyed to the session secondary partner name object and is maintained in a list to ensure that there is only one session established with the secondary partner.

To begin a session, the primary partner obtains an RPC binding handle (0x004377b0) from the secondary partner name object, as described in section 1.3.2. The primary partner uses the binding handle to send a **BuildContextW** call to the secondary partner using SRANK_PRIMARY. In the **BuildContextW** call, the primary partner passes its NetBIOS machine name (pszHostName) and contact identifier (CID) (pszUuidString), and the secondary partner's contact identifier (CID) (pszCalleeUuid). The primary partner also sends the session GUID (pszGuidIn), which will be returned in pszGuidOut when the session is accepted. In the BindVersionSet, the primary partner indicates that it supports both ASCII and Unicode method calls, that it supports version 1 of the next higher-layer protocol and version 5 of the third-layer protocol. (In this example, this is version 1 of the MSDTC Connection Manager: OleTx Multiplexing Protocol and version 5 of the MSDTC Connection Manager: OleTx Transaction Protocol, which is the current version at the level of Windows XP SP2, Windows Server 2003 SP1, or Windows Vista.) In the BindInfo (rguchBlob), the primary partner indicates that it supports PROT_IP_TCP (bit 0) and PROT_LRPC (bit 5). See section 2.2.7. The primary partner also passes a pointer to a PCONTEXT_HANDLE, into which it will receive the secondary partner PCONTEXT_HANDLE when the session is accepted.

Copyright © 2008 Microsoft Corporation.

Field	Value description
hRPC	RPC_BINDING_HANDLE=0x004377b0
sRank	SRANK_PRIMARY
BindVersionSet	dwMinLevelOne : 1
	dwMaxLevelOne : 2
	dwMinLevelTwo : 1
	dwMaxLevelTwo : 1
	dwMinLevelThree : 1
	dwMaxLevelThree : 5
pwszCalleeUuid	L"a3afb37b-f64a-4e6c-9017-f6a96ba6f166"
pwszHostName	L"Machine_1"
pwszUuidString	L"b51996ef-c434-4f79-a288-56efd302fc8e"
pwszGuidIn	L"a5acacb4-b766-4074-b45d-ade720d1d8e8"
pwszGuidOut [in_out]	L"00000000-0000-0000-0000-000000000000"
pBoundVersionSet [in_out]	dwLevelOneAccepted : 0
	dwLevelTwoAccepted : 0
	dwLevelThreeAccepted : 0
dwcbSizeOfBlob	dwcbSizeOfBlob: 8
rguchBlob	dwcbThisStruct : 8
	PROT_IP_TCP | PROT_LRPC
ppHandle [out]	*PPCONTEXT_HANDLE=0x00000000

When the secondary partner receives the **BuildContextW** call from the primary partner, the secondary partner attempts to locate an existing session object associated with the primary partner. If an existing session object is found, the secondary partner returns E_CM_SERVER_NOT_READY (0x80000123), which will occur if a previous session has not been completely torn down before a new session is begun.

Copyright © 2008 Microsoft Corporation.

If no existing session is found, the secondary partner will create a new session object with session GUID passed to it from the primary partner. The session object is keyed to the primary partner name object and is maintained in a list maintained by the secondary partner to ensure that one session is established with the primary partner.

To complete the session, the secondary partner obtains an RPC binding handle (0x001e7bd0) from the primary partner's name object, as described in section 1.3.2. The secondary partner uses the binding handle to send a **BuildContextW** message call to the primary partner using SRANK_SECONDARY. In the **BuildContextW** call to the primary partner, the secondary partner passes its NetBIOS machine name (pwszHostName) and contact identifier (CID) (pwszUuidString) and the primary partner's contact identifier (CID) (pwszCalleeUuid). The secondary partner also passes in the primary partner's session GUID (pwszGuidIn) from the initial call and a pointer to a PCONTEXT_HANDLE, which will be filled when the primary partner accepts the session.

Field	Value description
hRPC	RPC_BINDING_HANDLE=0x001e7bd0
sRank	SRANK_SECONDARY
BindVersionSet	dwMinLevelOne : 1
	dwMaxLevelOne : 2
	dwMinLevelTwo : 1
	dwMaxLevelTwo : 1
	dwMinLevelThree : 1
	dwMaxLevelThree : 5
pwszCalleeUuid	L"b51996ef-c434-4f79-a288-56efd302fc8e"
pwszHostName	L"Machine_2"
pwszUuidString	L"a3afb37b-f64a-4e6c-9017-f6a96ba6f166"
pwszGuidIn	L"a5acacb4-b766-4074-b45d-ade720d1d8e8"
pwszGuidOut [in_out]	L"00000000-0000-0000-0000-000000000000"
pBoundVersionSet [in_out]	dwLevelOneAccepted : 0
	dwLevelTwoAccepted : 0
	dwLevelThreeAccepted : 0
dwcbSizeOfBlob [in_out]	dwcbSizeOfBlob: 8

Copyright © 2008 Microsoft Corporation.

Field	Value description
rguchBlob	dwcbThisStruct : 8
	PROT_IP_TCP | PROT_LRPC
ppHandle [out]	*PPCONTEXT_HANDLE=0x00000000

When the **BuildContextW** call is received by the primary partner, the primary partner fills in the pwszGuidOut with the session GUID from pwszGuidIn, and will fill in the BoundVersionSet with its accepted values. The primary partner will also pass a reference pointer (0x00436e68) to the context object associated with its session object via the PPCONTEXT_HANDLE, and reply S_OK. Once the session is established, all future communication from the secondary partner will reference this PCONTEXT_HANDLE.

Field	Value description
pwszGuidOut [in_out]	L"a5acacb4-b766-4074-b45d-ade720d1d8e8"
pBoundVersionSet [in_out]	dwLevelOneAccepted : 2
	dwLevelTwoAccepted : 1
	dwLevelThreeAccepted : 5
ppHandle [out]	*PPCONTEXT_HANDLE=0x00436e68

When S_OK is returned to the secondary partner on its **BuildContextW** call, the secondary partner fills in the pszGuidOut with the session GUID from pszGuidIn and sets the accepted values for the BoundVersionSet. The secondary partner will also pass a reference pointer (0x0053b710) to the context object associated with its session object via the PPCONTEXT_HANDLE, and will reply S_OK. Once the session is established, all future communication from the primary partner will need to reference this PCONTEXT_HANDLE.

Field	Value description
pwszGuidOut [in_out]	L"a5acacb4-b766-4074-b45d-ade720d1d8e8"
pBoundVersionSet [in_out]	dwLevelOneAccepted : 2
	dwLevelTwoAccepted : 1
	dwLevelThreeAccepted : 5
ppHandle [out]	*PPCONTEXT_HANDLE=0x0053b710

At this point, a session has been established between the primary partner and the secondary partner. Either partner is now free to call **NegotiateResources** and initiate Connections.

Copyright © 2008 Microsoft Corporation.

4.2 Initiating a Session as Secondary Partner

In this example, the first partner is on Machine_1 with contact identifier (CID) (474cf518-d7ae-451f-a31f-caad29fa5e9f), and the second partner is on Machine_2 with contact identifier (CID) (a3afb37b-f64a-4e6c-9017-f6a96ba6f166). Therefore, the first partner assumes the role of the secondary partner, and the second partner assumes the role of the primary partner. This example assumes that the secondary partner does not have an existing session with the primary partner, as there is only one established session between any two partners.

Since this is a new session, the secondary partner will create a new session object. However, the secondary partner will not generate a session GUID, but will obtain the session GUID from the primary partner **BuildContextW** call. The session object is keyed to the primary partner's name object and is maintained in a list for the secondary partner to ensure that there is only one session established with the primary partner.

To begin a session, the secondary partner obtains an RPC binding handle (0x00227b88) from the primary partner's name object, as described in section 1.3.2. Since it is against protocol for the secondary partner to send the first **BuildContextW** call, the secondary partner uses the binding handle to send a **PokeW** call to the primary partner. In the **Poke** call, the secondary partner passes its NetBIOS machine name (pszHostName) and contact identifier (pszUuidString) and the primary partner contact identifier (pszCalleeUuid). In the BindInfo (rguchBlob), the secondary partner indicates that it supports PROT_IP_TCP (bit 0) and PROT_LRPC (bit 5). See section 2.2.7.

Field	Value description
hRPC	RPC_BINDING_HANDLE=0x00227b88
sRank	SRANK_SECONDARY
pwszCalleeUuid	L"a3afb37b-f64a-4e6c-9017-f6a96ba6f166"
pwszHostName	L"Machine_1"
pwszUuidString	L"474cf518-d7ae-451f-a31f-caad29fa5e9f"
dwcbSizeOfBlob	dwcbSizeOfBlob: 8
rguchBlob	dwcbThisStruct : 8
	PROT_IP_TCP | PROT_LRPC

When the primary partner receives the **Poke** call from the secondary partner, the primary partner will attempt to locate an existing session object associated with the secondary partner. If an existing session object is found, the primary partner returns E_CM_SERVER_NOT_READY (0x80000123), which will occur if a previous session has not been completely torn down before a new session is begun.

If no existing session is found, the primary partner will create a new session object and identify it with a newly generated session GUID. The session object is keyed to the secondary partner's name object and is maintained in a list for the primary partner to ensure that there is only one session established with the secondary partner. At this point, the primary partner replies S_OK to the **Poke** call from the secondary partner, and assumes the role of the primary partner.

As in the first example (see section 4.1), the primary partner obtains an RPC binding handle (0x004dae28) from the secondary partner's name object (see section 1.3.2) to begin a session. The

Copyright © 2008 Microsoft Corporation.

primary partner uses the binding handle to send a **BuildContextW** call to the secondary partner using SRANK_PRIMARY. In the **BuildContextW** call, the primary partner passes its NetBIOS machine name (pszHostName) and contact identifier (pszUuidString) and the secondary partner's contact identifier (pszCalleeUuid). The primary partner also sends the session GUID (pszGuidIn), which will be returned in pszGuidOut when the session is accepted. In the BindVersionSet, the primary partner indicates that it supports both ASCII and Unicode method calls, that it supports version 1 of the next higher-layer protocol and version 5 of the third-layer protocol. In the BindInfo (rguchBlob), the primary partner indicates that it supports PROT_IP_TCP (bit 0) and PROT_LRPC (bit 5). See section 2.2.7. The primary partner also passes a pointer to a PCONTEXT_HANDLE into which it will receive the secondary partner's PCONTEXT_HANDLE when the session is accepted.

Field	Value description
hRPC	RPC_BINDING_HANDLE=0x004dae28
sRank	SRANK_PRIMARY
BindVersionSet	dwMinLevelOne : 1
	dwMaxLevelOne : 2
	dwMinLevelTwo : 1
	dwMaxLevelTwo : 1
	dwMinLevelThree : 1
	dwMaxLevelThree : 5
pwszCalleeUuid	L"474cf518-d7ae-451f-a31f-caad29fa5e9f"
pwszHostName	L"Machine_2"
pwszUuidString	L"a3afb37b-f64a-4e6c-9017-f6a96ba6f166"
pwszGuidIn	L"79135638-e1c2-4fb5-9a47-6951d28e4d9c"
pwszGuidOut [in_out]	L"00000000-0000-0000-0000-000000000000"
pBoundVersionSet [in_out]	dwLevelOneAccepted : 0
	dwLevelTwoAccepted : 0
	dwLevelThreeAccepted : 0
dwcbSizeOfBlob	dwcbSizeOfBlob: 8
rguchBlob	dwcbThisStruct : 8

Copyright © 2008 Microsoft Corporation.

Field	Value description
	PROT_IP_TCP | PROT_LRPC
ppHandle [out]	*PPCONTEXT_HANDLE=0x00000000

When the secondary partner receives the **BuildContextW** call from the primary partner, the secondary partner will locate the existing session object associated with the primary partner, and will copy in the session GUID passed to it from the primary partner.

Because the primary partner has specified that it supports both the ASCII and Unicode method calls (dwMaxLevelOne = 2), the secondary partner sends a **BuildContextW** message call to the primary partner using SRANK_SECONDARY. In the **BuildContextW** call to the primary partner, the secondary partner passes its NetBIOS machine name (pwszHostName) and contact identifier (CID) (pwszUuidString), and the primary partner contact identifier (CID) (pwszCalleeUuid). The secondary partner also passes in the primary partner's session GUID (pwszGuidIn) from the initial call. The secondary partner also passes a pointer to a PCONTEXT_HANDLE, which will be filled when the primary partner accepts the session.

Field	Value description
hRPC	RPC_BINDING_HANDLE=0x00227b88
sRank	SRANK_SECONDARY
BindVersionSet	dwMinLevelOne : 1
	dwMaxLevelOne : 2
	dwMinLevelTwo : 1
	dwMaxLevelTwo : 1
	dwMinLevelThree : 1
	dwMaxLevelThree : 5
pwszCalleeUuid	L"a3afb37b-f64a-4e6c-9017-f6a96ba6f166"
pwszHostName	L"Machine_1"
pwszUuidString	L"474cf518-d7ae-451f-a31f-caad29fa5e9f"
pwszGuidIn	L"79135638-e1c2-4fb5-9a47-6951d28e4d9c"
pwszGuidOut [in_out]	L"00000000-0000-0000-0000-000000000000"
pBoundVersionSet [in_out]	dwLevelOneAccepted : 0

Copyright © 2008 Microsoft Corporation.

Field	Value description
	dwLevelTwoAccepted : 0
	dwLevelThreeAccepted : 0
dwcbSizeOfBlob	dwcbSizeOfBlob: 8
rguchBlob	dwcbThisStruct : 8
	PROT_IP_TCP | PROT_LRPC
ppHandle [out]	*PPCONTEXT_HANDLE=0x00000000

When the **BuildContextW** call is received by the primary partner, the primary partner fills in the pwszGuidOut with the session GUID from pwszGuidIn, and will fill in the BoundVersionSet with its accepted values. The primary partner will also pass a reference pointer (0x0012af48) to the context object associated with its session object via the PPCONTEXT_HANDLE, and replies S_OK. Once the session is established, all future communication from the secondary partner will reference this PCONTEXT_HANDLE.

Field	Value description
pwszGuidOut [in_out]	L"79135638-e1c2-4fb5-9a47-6951d28e4d9c"
pBoundVersionSet [in_out]	dwLevelOneAccepted : 2
	dwLevelTwoAccepted : 1
	dwLevelThreeAccepted : 5
ppHandle [out]	*PPCONTEXT_HANDLE=0x0012af48

When S_OK is returned to the secondary partner on its **BuildContextW** call, the secondary partner fills in the pszGuidOut with the session GUID from pszGuidIn and sets the accepted values for the BoundVersionSet. The secondary partner will also pass a reference pointer (0x00bf90e0) to the context object associated with its session object via the PPCONTEXT_HANDLE, and reply S_OK. Once the session is established, all future communication from the primary partner will need to reference this PCONTEXT_HANDLE.

Field	Value description
pwszGuidOut [in_out]	L"79135638-e1c2-4fb5-9a47-6951d28e4d9c"
pBoundVersionSet [in_out]	dwLevelOneAccepted : 2
	dwLevelTwoAccepted : 1

Copyright © 2008 Microsoft Corporation.

Field	Value description
	dwLevelThreeAccepted : 5
ppHandle [out]	*PPCONTEXT_HANDLE=0x005f90e0

At this point, a session has been established between the primary partner and the secondary partner. Either partner is now free to call **NegotiateResources** and initiate Connections.

4.3 Negotiating Connection Resources

After a session is established, each partner needs to respond to requests from MSDTC Connection Manager: OleTx Multiplexing Protocol to negotiate resources with its partner.

In this example, the first partner requests one hundred (100) connection resources from the second partner. The first partner will pass in the PCONTEXT_HANDLE that it received from its **BuildContext** (or **BuildContextW**) call to the second partner and the ResourceType for the connection resources (0 in this example).

Field	Value description
phContext	PCONTEXT_HANDLE=0x0053b710
ResourceType	0
dwcRequested	100
pdwcAccepted [in_out]	0

When the second partner receives the **NegotiateResources** call, it will attempt to allocate sufficient resources to support the 100 concurrent Connections requested. If successful, the second partner will return S_OK and indicate that all 100 concurrent Connection Resources have been allocated.

Field	Value description
pdwcAccepted [in_out]	100

When the first partner receives the S_OK from the second partner, the first partner is now ready to begin establishing Connections with the second partner.

4.4 Terminating a Session

Terminating a session follows a similar protocol handshake as that of establishing a session (see section 4.1).

A session is terminated by the primary partner sending a **TearDownContext** call to the secondary partner. In response, the secondary partner sends a **TearDownContext** call to the primary partner. When the primary partner returns success to the **TearDownContext** call from the secondary partner, the secondary partner returns success to the primary partner's **TearDownContext** call. Since the first **TearDownContext** call in the sequence originates from the primary partner, the secondary partner is only allowed to initiate teardown of a session with the primary partner by calling **BeginTearDown**, which instructs the primary partner to send a **TearDownContext** call to the secondary partner.

Copyright © 2008 Microsoft Corporation.

4.4.1 Terminating a Session by a Primary Partner

A primary partner terminates a session by sending a **TearDownContext** call to the secondary partner, passing a pointer to the PCONTEXT_HANDLE given to it from the secondary partner, its SESSION_RANK (that is, SRANK_PRIMARY), and a reason for tearing down the session; in this example, the TEAR_DOWN_TYPE is TT_FORCE.

Field	Value description
pphContext [in_out]	*PPCONTEXT_HANDLE=0x0053b710
sRank	SRANK_PRIMARY
TearDownType	TT_FORCE

When the secondary partner receives the **TearDownContext** call, it will send a **TearDownContext** call to the primary partner, passing a pointer to the PCONTEXT_HANDLE passed to it from the primary partner, its SESSION_RANK (that is, SRANK_SECONDARY), and copy the TEAR_DOWN_TYPE from the incoming call (that is, TT_FORCE).

Field	Value description
pphContext [in_out]	*PPCONTEXT_HANDLE=0x00436e68
sRank	SRANK_SECONDARY
TearDownType	TT_FORCE

When the primary partner receives the **TearDownContext** request, it will delete its PCONTEXT_HANDLE and null out pphContext. Any negotiated resources will be released, and it will reply S_OK.

Field	Value description
pphContext [in_out]	*PPCONTEXT_HANDLE=0x00000000

When the secondary partner receives S_OK on the **TearDownContext** call, it will delete its PCONTEXT_HANDLE and null out pphContext. Any negotiated resources will be released, and it will reply S_OK.

Field	Value description
pphContext [in_out]	*PPCONTEXT_HANDLE=0x00000000

The session has now been terminated, and no further messages will be sent.

4.4.2 Terminating a Session by a Secondary Partner

In this example, the secondary partner initiates the session termination process by sending a **BeginTearDown** call to the primary partner, passing the primary partner's PCONTEXT_HANDLE and the reason for the tear-down request; in this example, the TEAR_DOWN_TYPE is TT_FORCE.

Copyright © 2008 Microsoft Corporation.

Field	Value description
phContext	PCONTEXT_HANDLE=0x005f90e0
TearDownType	TT_FORCE

When the primary partner receives the **BeginTearDown** call, it will send a **TearDownContext** call to the secondary partner, passing a pointer to the secondary partner PCONTEXT_HANDLE, its SESSION_RANK (that is, SRANK_PRIMARY), and a reason for tearing down the session sent to it in the **BeginTearDown** call (that is, TT_FORCE).

Field	Value description
pphContext [in_out]	*PPCONTEXT_HANDLE=0x0012af48
sRank	SRANK_PRIMARY
TearDownType	TT_FORCE

When the secondary partner receives the **TearDownContext** call, it will send a **TearDownContext** call to the primary partner, passing a pointer to the PCONTEXT_HANDLE passed to it from the primary partner, its SESSION_RANK (that is, SRANK_SECONDARY), and copy the TEAR_DOWN_TYPE from the incoming call (that is, TT_FORCE).

Field	Value description
pphContext [in_out]	*PPCONTEXT_HANDLE=0x005f90e0
sRank	SRANK_SECONDARY
TearDownType	TT_FORCE

When the primary partner receives the **TearDownContext** request, it will delete its PCONTEXT_HANDLE and null out pphContext. Any negotiated resources will be released, and it will reply S_OK.

Field	Value description
pphContext [in_out]	*PPCONTEXT_HANDLE=0x00000000

When the secondary partner receives S_OK on the **TearDownContext** call, it will delete its PCONTEXT_HANDLE and null out pphContext. Any negotiated resources will be released, and it will reply S_OK.

Field	Value description
pphContext [in_out]	*PPCONTEXT_HANDLE=0x00000000

The session has now been terminated, and no further messages will be sent.

Copyright © 2008 Microsoft Corporation.

5 Security

The following sections specify security considerations for implementers of the MSDTC Connection Manager: OleTx Transports Protocol.

5.1 Security Considerations for Implementers

For security considerations for both unauthenticated RPC and authenticated RPC used in this protocol, see [MS-RPCE].

The client MAY fail over to unauthenticated RPC when authenticated RPC fails for backward compatibility. The unauthenticated RPC is not as secure as authenticated RPC; the client SHOULD either audit or support this automatic failover only when it is explicitly specified.<34>

5.2 Index of Security Parameters

None.

Copyright © 2008 Microsoft Corporation.

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided below.

```
import "ms-dtyp.idl";
[
    uuid (906B0CE0-C70B-1067-B317-00DD010662DA),
    version(1.0),
    pointer_default(ref)
]
interface IXnRemote
{

#ifdef __midl
#if __midl >= 700
#define use_string_range
#endif
#endif

#ifndef MAX_COMPUTERNAME_LENGTH
#define MAX_COMPUTERNAME_LENGTH 15
#endif

    typedef enum _TearDownType
    {
        TT_FORCE = 0,
        TT_PROBLEM = 2,
    } TEARDOWN_TYPE;

    typedef enum _SessionRank
    {
        SRANK_PRIMARY = 1,
        SRANK_SECONDARY = 2
    } SESSION_RANK;

    typedef struct _BindVersionSet
    {
        DWORD   dwMinLevelOne;
        DWORD   dwMaxLevelOne;
        DWORD   dwMinLevelTwo;
        DWORD   dwMaxLevelTwo;
        DWORD   dwMinLevelThree;
        DWORD   dwMaxLevelThree;
    } BIND_VERSION_SET;

    typedef struct _BoundVersionSet
    {
        DWORD   dwLevelOneAccepted;
        DWORD   dwLevelTwoAccepted;
        DWORD   dwLevelThreeAccepted;
    } BOUND_VERSION_SET;

    typedef unsigned long COM_PROTOCOL;


    typedef struct _BindInfoBlob
    {
```


Copyright © 2008 Microsoft Corporation.

```
        DWORD       dwcbThisStruct;

        COM_PROTOCOL grbitComProtocols;
    } BIND_INFO_BLOB;

    typedef [context_handle] void * PCONTEXT_HANDLE;
    typedef [ref] PCONTEXT_HANDLE * PPCONTEXT_HANDLE;


#define GUID_LENGTH 37

#ifdef use_string_range
    HRESULT Poke (
      [in] handle_t hBinding,
      [in] SESSION_RANK sRank,
      [in, string, range(GUID_LENGTH, GUID_LENGTH)]
           unsigned char  pszCalleeUuid[],
      [in, string, range(1, MAX_COMPUTERNAME_LENGTH+1)]
           unsigned char pszHostName[],
      [in, string, range(GUID_LENGTH, GUID_LENGTH)]
           unsigned char pszUuidString[],
      [in, range(sizeof(BIND_INFO_BLOB),sizeof(BIND_INFO_BLOB))]
           DWORD   dwcbSizeOfBlob,
      [in, size_is (dwcbSizeOfBlob)] unsigned char rguchBlob[]);
#else
     HRESULT Poke (
[in] handle_t hBinding,
      [in] SESSION_RANK sRank,
      [in, string] unsigned char  pszCalleeUuid[],
      [in, string] unsigned char pszHostName[],
      [in, string] unsigned char pszUuidString[],
      [in, range(sizeof(BIND_INFO_BLOB),sizeof(BIND_INFO_BLOB))]
            DWORD   dwcbSizeOfBlob,
      [in, size_is (dwcbSizeOfBlob)] unsigned char rguchBlob[]);
#endif

#ifdef use_string_range
    HRESULT BuildContext (
      [in] handle_t hBinding,
      [in] SESSION_RANK sRank,
      [in] BIND_VERSION_SET BindVersionSet,
      [in, string, range(GUID_LENGTH, GUID_LENGTH)]
           unsigned char  pszCalleeUuid[],
      [in, string, range(1, MAX_COMPUTERNAME_LENGTH+1)]
           unsigned char pszHostName[],
      [in, string, range(GUID_LENGTH, GUID_LENGTH)]
           unsigned char pszUuidString[],
      [in, string, range(GUID_LENGTH, GUID_LENGTH)]
           unsigned char pszGuidIn[],
      [in, out, string, range(GUID_LENGTH, GUID_LENGTH)]
           unsigned char pszGuidOut[],
      [in, out] BOUND_VERSION_SET *   pBoundVersionSet,
      [in, range(sizeof(BIND_INFO_BLOB), sizeof(BIND_INFO_BLOB))]
           DWORD dwcbSizeOfBlob,
      [in, size_is (dwcbSizeOfBlob)] unsigned char rguchBlob[],
      [out] PPCONTEXT_HANDLE ppHandle);
#else
    HRESULT BuildContext (
```


Copyright © 2008 Microsoft Corporation.

```
      [in] handle_t hBinding,
      [in] SESSION_RANK sRank,
      [in] BIND_VERSION_SET BindVersionSet,
      [in, string] unsigned char  pszCalleeUuid[],
      [in, string] unsigned char pszHostName[],
      [in, string] unsigned char pszUuidString[],
      [in, string] unsigned char pszGuidIn[],
      [in, out, string] unsigned char pszGuidOut[],
      [in, out] BOUND_VERSION_SET *   pBoundVersionSet,
      [in, range(sizeof(BIND_INFO_BLOB), sizeof(BIND_INFO_BLOB))]
            DWORD dwcbSizeOfBlob,
      [in, size_is (dwcbSizeOfBlob)] unsigned char rguchBlob[],
      [out] PPCONTEXT_HANDLE ppHandle);
#endif

    HRESULT NegotiateResources (
      [in] PCONTEXT_HANDLE phContext,
      [in] DWORD resourceType,
      [in] DWORD  dwcRequested,
      [in,out] DWORD * pdwcAccepted);

    HRESULT SendReceive (
      [in] PCONTEXT_HANDLE phContext,
      [in, range(1, 4095)] DWORD  dwcMessages,
      [in, range(32, 0x14000)] DWORD  dwcbSizeOfBoxCar,
      [in, size_is (dwcbSizeOfBoxCar)]
           unsigned char rguchBoxCar[]);

    HRESULT TearDownContext (
      [in, out] PPCONTEXT_HANDLE contextHandle,
      [in] SESSION_RANK sRank,
      [in] TEARDOWN_TYPE tearDownType);

    HRESULT BeginTearDown (
      [in] PCONTEXT_HANDLE contextHandle,
      [in] TEARDOWN_TYPE tearDownType);

#ifdef use_string_range
    HRESULT PokeW (
      [in] handle_t hBinding,
      [in] SESSION_RANK sRank,
      [in, string, range(GUID_LENGTH, GUID_LENGTH)]
           wchar_t pwszCalleeUuid[],
      [in, string, range(1, MAX_COMPUTERNAME_LENGTH+1)]
           wchar_t pwszHostName[],
      [in, string, range(GUID_LENGTH, GUID_LENGTH)]
           wchar_t pwszUuidString[],
      [in, range(sizeof(BIND_INFO_BLOB), sizeof(BIND_INFO_BLOB))]
           DWORD dwcbSizeOfBlob,
      [in, size_is (dwcbSizeOfBlob)] unsigned char rguchBlob[]);
#else
    HRESULT PokeW (
      [in] handle_t hBinding,
      [in] SESSION_RANK sRank,
      [in, string] wchar_t pwszCalleeUuid[],
      [in, string] wchar_t pwszHostName[],
      [in, string] wchar_t pwszUuidString[],
      [in, range(sizeof(BIND_INFO_BLOB), sizeof(BIND_INFO_BLOB))]
```


Copyright © 2008 Microsoft Corporation.

```
            DWORD dwcbSizeOfBlob,
      [in, size_is (dwcbSizeOfBlob)] unsigned char rguchBlob[]);
#endif

#ifdef use_string_range
    HRESULT BuildContextW (
      [in] handle_t hBinding,
      [in] SESSION_RANK sRank,
      [in] BIND_VERSION_SET BindVersionSet,
      [in, string, range(GUID_LENGTH, GUID_LENGTH)]
           wchar_t  pwszCalleeUuid[],
      [in, string, range(1, MAX_COMPUTERNAME_LENGTH+1)]
           wchar_t pwszHostName[],
      [in, string, range(GUID_LENGTH, GUID_LENGTH)]
           wchar_t pwszUuidString[],
      [in, string, range(GUID_LENGTH, GUID_LENGTH)]
           wchar_t pwszGuidIn[],
      [in,out, string, range(GUID_LENGTH, GUID_LENGTH)]
           wchar_t pwszGuidOut[],
      [in, out] BOUND_VERSION_SET *pBoundVersionSet,
      [in, range(sizeof(BIND_INFO_BLOB), sizeof(BIND_INFO_BLOB))]
           DWORD dwcbSizeOfBlob,
      [in, size_is (dwcbSizeOfBlob)] unsigned char rguchBlob[],
       [out] PPCONTEXT_HANDLE ppHandle);
#else
    HRESULT BuildContextW (
     [in] handle_t hBinding,
     [in] SESSION_RANK sRank,
     [in] BIND_VERSION_SET BindVersionSet,
     [in, string] wchar_t  pwszCalleeUuid[],
     [in, string] wchar_t pwszHostName[],
     [in, string] wchar_t pwszUuidString[],
     [in, string] wchar_t pwszGuidIn[],
     [in,out, string] wchar_t pwszGuidOut[],
     [in, out] BOUND_VERSION_SET *pBoundVersionSet,
     [in, range(sizeof(BIND_INFO_BLOB), sizeof(BIND_INFO_BLOB))]
           DWORD dwcbSizeOfBlob,
     [in, size_is (dwcbSizeOfBlob)] unsigned char rguchBlob[],
     [out] PPCONTEXT_HANDLE ppHandle);
#endif

#undef GUID_LENGTH
}
```


Copyright © 2008 Microsoft Corporation.

7 Appendix B: Windows Behavior

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

- Windows NT 4.0 Option Pack implements version 2.0 of the protocol. For the remainder of this section, "Windows NT" refers to Windows NT 4.0 Option Pack.
- Windows 2000 implements version 2.0 of the protocol.
- Windows XP implements the client side of version 2.0 of the protocol.
- Windows Server 2003 implements version 2.0 of the protocol.
- Windows Vista does not support the protocol. Any attempts to use these methods on Windows Vista cause a nonzero error code to be returned.
- Windows Server 2008 supports version 3.0 of the protocol, except as noted below. Windows Vista SP1 will support version 3.0 of the protocol, except as noted below.

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 1.7: Windows implementations of MSDTC Connection Manager: OleTx Transports Protocol use by default the SPNEGO Security Provider described in [MS-SPNG], which allows for in-band negotiation of a Security Provider package.

<2> Section 2.1.1: Windows NT 4.0 Option Pack, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 support "ncacn_ip_tcp" by default, but can be configured to support any or all of the "ncacn_ip_udp", "ncacn_spx", and "ncacn_nb_nb" protocols with the exception that Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 do not support "ncacn_spx."

<3> Section 2.1.3: On Windows, the MSDTC Connection Manager: OleTx Transports Protocol supports two types of connections: secure and unsecure. For the unsecure connection type, the default RPC security settings are used, and no attempt is made to change or alter them in any way.

For the secure connection type, the MSDTC Connection Manager: OleTx Transports Protocol configures its RPC binding handles to support authentication to the remote server, and to do mutual authentication with the remote server. The expected server principal name varies depending on the nature of the connection. The security settings can be configured but, by default, they are as follows:

- **Authentication level** defaults to RPC_C_AUTHN_LEVEL_PKT_PRIVACY.
- **Authentication Service (AS)** defaults to RPC_C_AUTHN_GSS_NEGOTIATE.
- Quality-of-Service capabilities default to RPC_C_QOS_CAPABILITIES_MUTUAL_AUTH.
- Quality-of-Service identity tracking defaults to RPC_C_QOS_IDENTITY_STATIC.

The quality-of-service impersonation type is always set to RPC_C_IMP_LEVEL_IDENTIFY, and the Authentication Service (AS) is always set to RPC_C_AUTHZ_NONE; these settings are not configurable.

Copyright © 2008 Microsoft Corporation.

Regardless of the configuration settings, the MSDTC Connection Manager: OleTx Transports Protocol starts out by configuring new unresolved RPC binding handles with the following security settings:

- Authentication level is set to the configured authentication level.
- dy (AS) is set to RPC_C_AUTHN_WINNT.
- Security quality-of-service settings are set to their RPC defaults.

All other RPC security settings are left unchanged. These settings are used to contact the RPC endpoint mapper and resolve the binding handle. They are also used to query the remote partner server principal name. If the query fails with RPC_S_SERVER_TOO_BUSY or RPC_S_UNKNOWN_AUTHN_SERVICE, the MSDTC Connection Manager: OleTx Transports Protocol assumes that the remote partner did not configure security; it does not make any further configuration changes to the binding handle and considers the binding handle establishment to be a success. If the query fails for some other reason, it considers handle establishment to be a failure.

If the query succeeds, the MSDTC Connection Manager: OleTx Transports Protocol reconfigures the binding handle with the configured security settings and the expected server principal name. The binding handle is then used to make the necessary remote procedure call (RPC).

Different versions of Windows support different modes for using these connection types. The simplest mode is "unsecure", in which only unsecure connections are used.

In the "allow fallback if necessary" mode, the MSDTC Connection Manager: OleTx Transports Protocol first attempts to establish a connection in secure mode; failing this, it attempts to establish an unsecure connection.

In the "mutual authentication required" and "incoming authentication required" modes, the MSDTC Connection Manager: OleTx Transports Protocol initially attempts to establish a secure connection. In "mutual authentication required" mode, if the initial attempt fails, the MSDTC Connection Manager: OleTx Transports Protocol refuses to communicate with the other partner. In "incoming authentication required" mode, the MSDTC Connection Manager: OleTx Transports Protocol will fall back to an unsecure connection. In both of these modes, the MSDTC Connection Manager: OleTx Transports Protocol requires that incoming connections be authenticated; remote procedure call (RPC) from unauthenticated clients are rejected.

The "unsecure" mode is supported by Windows NT 4.0 Option Pack, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. The "allow fallback if necessary" mode is supported by Windows Server 2003. The "mutual authentication required" and "incoming authentication required" modes are supported by Windows XP SP2 or later, Windows Server 2003 SP1 or later, Windows Vista, and Windows Server 2008.

<4> Section 2.2.7: Windows uses this flag to indicate support for the "ncalrpc" RPC protocol sequence.

<5> Section 3.2.2.1: Windows sets the default value of this timer to 1 minute.

<6> Section 3.2.2.2: Windows sets the default value of this timer to 10 seconds.

<7> Section 3.2.3: On Windows, the MSDTC Connection Manager: OleTx Transports Protocol client and server Security Provider security settings are configured through specific keys in the Windows registry.

<8> Section 3.2.4: Windows NT 4.0 Option Pack and Windows 2000 do not indicate to the RPC runtime that it is to perform such a check.

Copyright © 2008 Microsoft Corporation.

<9> Section 3.2.4.1: Windows does not call the **Poke** method on a secondary partner. Windows returns from the **Poke** method before the recursive call has completed.

<10> Section 3.2.4.1: Windows returns 0x80000123 (E_CM_SERVER_NOT_READY).

Name	Value
E_CM_SERVER_NOT_READY	0x80000123

<11> Section 3.2.4.1: Windows returns as soon as the session object has been initialized and added to the session table regardless of the eventual outcome of the entire operation.

<12> Section 3.2.4.1: Windows retries the nested call when the call fails, unless it fails with 0x80000172.

Name	Value
E_CM_VERSION_SET_NOTSUPPORTED	0x80000172

<13> Section 3.2.4.2.1: Windows returns 0x80000123 (E_CM_SERVER_NOT_READY).

Name	Value
E_CM_SERVER_NOT_READY	0x80000123

<14> Section 3.2.4.2.1: Windows retries the nested call.

<15> Section 3.2.4.2.2: Windows returns 0x80000120 (E_CM_SESSION_DOWN).

Name	Value
E_CM_SESSION_DOWN	0x80000120

<16> Section 3.2.4.3: The value of pdwcAccepted is smaller than the value of dwcRequested if the partner was incapable of allocating all of the requested resources.

<17> Section 3.2.4.3: If the context handle is invalid, Windows returns an E_INVALID_CONTEXT_HANDLE error code. If the session object is in the Requesting Teardown or Teardown state, Windows returns a 0x80000119 (E_CM_TEARING_DOWN) error code. Otherwise, if the session object is not in the Active state, Windows returns a 0x80000123 (E_CM_SERVER_NOT_READY) error code.

<18> Section 3.2.4.3: Windows returns E_INVALIDARG.

<19> Section 3.2.4.4: Windows returns one of the following error codes.

Value	Meaning
E_CM_TEARING_DOWN 0x80000119	The session object is in the Requesting Teardown or Teardown state.
E_CM_SERVER_NOT_READY 0x80000123	The session object is not in the Active state.
E_INVALID_CONTEXT_HANDLE	The context handle is invalid.

Copyright © 2008 Microsoft Corporation.

Value	Meaning
0x80000128	

<20> Section 3.2.4.5: If the context handle is invalid, Windows returns an E_INVALID_CONTEXT_HANDLE error code. If the session object is not in the Active state, Windows returns a 0x80000123 (E_CM_SERVER_NOT_READY) error code.

<21> Section 3.2.4.5.2: Windows performs these actions asynchronously.

<22> Section 3.2.4.6: If the context handle is invalid, Windows returns an E_INVALID_CONTEXT_HANDLE error code. If the session object is not in the Active state, Windows returns a 0x80000123 (E_CM_SERVER_NOT_READY) error code.

<23> Section 3.2.4.6: Windows performs these actions asynchronously.

<24> Section 3.2.5.1: Windows does not cancel outstanding calls to **BuildContext** or **BuildContextW**.

<25> Section 3.2.5.1: Windows returns 0x80000124 (E_CM_S_TIMEDOUT).

Name	Value
E_CM_S_TIMEDOUT	0x80000124

<26> Section 3.2.5.2: Windows does not cancel outstanding calls to **TearDownContext**.

<27> Section 3.2.5.2: Windows returns 0x80040005 (E_FAIL).

Name	Value
E_FAIL	0x80040005

<28> Section 3.2.6.1: Windows handles the context handle rundown asynchronously with respect to the normal operation of the protocol.

<29> Section 3.3.4: The strict Network Data Representation (NDR) data consistency check is indicated to the RPC runtime on Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<30> Section 3.3.6.1.1: Windows returns 0x80000123 (E_CM_SERVER_NOT_READY).

Name	Value
E_CM_SERVER_NOT_READY	0x80000123

<31> Section 3.3.6.1.1: Windows retries the nested call.

<32> Section 3.3.6.1.2: Windows returns 0x80000123 (E_CM_SERVER_NOT_READY).

Name	Value
E_CM_SERVER_NOT_READY	0x80000123

<33> Section 3.3.6.1.2: Windows retries the nested call.

Copyright © 2008 Microsoft Corporation.

<34> Section 5.1: The usage of unauthenticated RPC is supported by Windows NT 4.0 Option Pack, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. The usage of authenticated RPC is supported and is the default on Windows XP SP2 and later, Windows Server 2003, Windows Vista, and Windows Server 2008; furthermore, these systems do not allow fallback to unauthenticated RPC by default, but can be configured to do so.

Copyright © 2008 Microsoft Corporation.

8 Index

Copyright © 2008 Microsoft Corporation.

Copyright © 2008 Microsoft Corporation.