

[MS-DCOM]: Distributed Component Object Model (DCOM) Remote 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
03/14/2007	1.0		Version 1.0 release
04/10/2007	1.1		Version 1.1 release
05/18/2007	1.2		Version 1.2 release
06/08/2007	1.2.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
07/10/2007	2.0	Major	Converted to unified format; updated technical content.
08/17/2007	3.0	Major	Updated and revised the technical content.
09/21/2007	4.0	Major	Added an interface.
10/26/2007	4.0.1	Editorial	Revised and edited the technical content.
01/25/2008	4.0.2	Editorial	Revised and edited the technical content.

Table of Contents

1	Introduction	7
1.1	Glossary	7
1.2	References	8
1.2.1	Normative References	8
1.2.2	Informative References	8
1.3	Protocol Overview (Synopsis)	9
1.3.1	Activation	11
1.3.2	Object References	12
1.3.3	Object Exporter.....	12
1.3.4	ORPC Calls.....	12
1.3.5	Causality Identifiers	13
1.3.6	Reference Counts	14
1.3.7	Object Resolver Service.....	14
1.4	Relationship to Other Protocols	14
1.5	Prerequisites/Preconditions	14
1.6	Applicability Statement	15
1.7	Versioning and Capability Negotiation	15
1.8	Vendor-Extensible Fields	16
1.9	Standards Assignments.....	16
2	Messages	19
2.1	Transport	19
2.2	Message Syntax	20
2.2.1	Common Data Types.....	20
2.2.1.1	OID.....	20
2.2.1.2	SETID.....	20
2.2.1.3	HRESULT	20
2.2.1.4	error_status_t	20
2.2.1.5	GUID.....	20
2.2.1.6	CID	21
2.2.1.7	CLSID.....	21
2.2.1.8	IID	21
2.2.1.9	IPID	21
2.2.1.10	OXID.....	21
2.2.1.11	COMVERSION	22
2.2.1.12	object IDL Attribute.....	22
2.2.1.13	ORPCTHIS and ORPCTHAT	23
2.2.1.13.1	ORPC_EXTENT	23
2.2.1.13.2	ORPC_EXTENT_ARRAY	23
2.2.1.13.3	ORPCTHIS	23
2.2.1.13.4	ORPCTHAT.....	24
2.2.1.14	MInterfacePointer.....	24
2.2.1.15	PMInterfacePointerInternal	24
2.2.1.16	iid_is IDL Attribute	25
2.2.1.17	OBJREF.....	25
2.2.1.17.1	STDOBJREF	26
2.2.1.17.2	STDOBJREF (Packet Version)	27
2.2.1.17.3	STDOBJREF (IDL Version).....	28
2.2.1.17.4	OBJREF_STANDARD.....	28
2.2.1.17.5	OBJREF_HANDLER	29
2.2.1.17.6	OBJREF_CUSTOM	30
2.2.1.17.7	OBJREF_EXTENDED	31

2.2.1.17.8 DATAELEMENT	32
2.2.1.18 DUALSTRINGARRAY	33
2.2.1.18.1 DUALSTRINGARRAY (Packet Version)	34
2.2.1.18.2 DUALSTRINGARRAY (IDL Version)	34
2.2.1.18.3 STRINGBINDING	35
2.2.1.18.4 SECURITYBINDING	36
2.2.1.19 Context	36
2.2.1.19.1 PROPMARSHALHEADER	38
2.2.1.20 ORPC Extensions.....	40
2.2.1.20.1 Error Information ORPC Extension	40
2.2.1.20.2 Custom-Marshaled Error Information Format	40
2.2.1.20.3 ErrorInfoString	42
2.2.1.20.4 Context ORPC Extension.....	42
2.2.1.20.5 EntryHeader	44
2.2.1.21 Activation Properties BLOB	45
2.2.1.21.1 CustomHeader	45
2.2.1.21.2 Activation Properties	46
2.2.1.21.2.1 InstantiationInfoData	46
2.2.1.21.2.2 SpecialPropertiesData.....	48
2.2.1.21.2.3 InstanceInfoData	49
2.2.1.21.2.4 ScmRequestInfoData.....	49
2.2.1.21.2.4.1 customREMOTE_REQUEST_SCM_INFO	49
2.2.1.21.2.5 ActivationContextInfoData	50
2.2.1.21.2.6 LocationInfoData	50
2.2.1.21.2.7 SecurityInfoData.....	51
2.2.1.21.2.7.1 COSERVERINFO	51
2.2.1.21.2.8 ScmReplyInfoData	52
2.2.1.21.2.8.1 customREMOTE_REPLY_SCM_INFO.....	52
2.2.1.21.2.9 PropsOutInfo	53
2.2.1.22 REMINTERFACEREF	53
2.2.1.23 REMQIRERESULT	54
2.2.1.24 PREMQIRERESULT	54
2.2.1.25 REFIID	54
2.2.1.26 Constant Definitions	54
2.2.1.26.1 RPC Authentication Level Constants	54
2.2.1.26.2 RPC Authentication Service Constants	55
2.2.1.26.3 IDL Range Constants	55
3 Protocol Details	56
3.1 Client Details.....	56
3.1.1 Abstract Data Model.....	56
3.1.2 Timers	57
3.1.3 Initialization.....	57
3.1.4 Higher-Layer Triggered Events	57
3.1.4.1 Creating Object References	57
3.1.4.1.1 Activation	58
3.1.4.1.1.1 Determining RPC Binding Information for Activation	58
3.1.4.1.1.2 Issuing the Activation Request.....	59
3.1.4.1.1.3 Updating the Client OXID Table after Activation	61
3.1.4.1.2 Unmarshaling an Object Reference	61
3.1.4.1.2.1 Determining RPC Binding Information for OXID Resolution	61
3.1.4.1.2.2 Issuing the OXID Resolution Request	62
3.1.4.1.2.3 Updating Client Tables after Unmarshaling	63
3.1.4.1.2.3.1 Updating the OXID Table after Unmarshaling	63
3.1.4.1.2.3.2 Updating the OID/IPID/Resolver Tables after Unmarshaling	64

3.1.4.2	ORPC Invocations	65
3.1.4.3	Marshaling an Object Reference	66
3.1.4.4	Managing Object Lifetime	67
3.1.4.4.1	Requesting Reference Counts on an Interface	67
3.1.4.4.2	Releasing Reference Counts on an Interface	68
3.1.4.4.3	Acquiring Additional Interfaces on the Object	68
3.1.5	Message Processing Events and Sequencing Rules	69
3.1.6	Timer Events	69
3.1.6.1	Pinging	69
3.1.7	Other Local Events	70
3.2	Object Server Details	70
3.2.1	Object Exporter Details	71
3.2.1.1	Abstract Data Model	71
3.2.1.2	Timers	72
3.2.1.3	Initialization	72
3.2.1.4	Higher-Layer Triggered Events	72
3.2.1.5	Message Processing Events and Sequencing Rules	72
3.2.1.5.1	Marshaling an Object	72
3.2.1.5.2	Marshaling an Object Reference	74
3.2.1.5.3	Unmarshaling an Object Reference	74
3.2.1.5.4	ORPC Invocations	75
3.2.1.5.5	Lazy Protocol Registration	76
3.2.1.5.6	IRemUnknown Interface	76
3.2.1.5.6.1	IRemUnknown Methods	76
3.2.1.5.6.1.1	IRemUnknown::RemQueryInterface (Opnum 3)	77
3.2.1.5.6.1.2	IRemUnknown::RemAddRef (Opnum 4)	78
3.2.1.5.6.1.3	IRemUnknown::RemRelease (Opnum 5)	79
3.2.1.5.7	IRemUnknown2 Interface	80
3.2.1.5.7.1	IRemUnknown2 Methods	80
3.2.1.5.7.1.1	IRemUnknown2::RemQueryInterface2 (Opnum 6)	80
3.2.1.5.8	IUnknown Interface	81
3.2.1.6	Timer Events	81
3.2.1.6.1	Pinging	81
3.2.1.6.2	Object Reclamation	81
3.2.1.7	Other Local Events	82
3.2.2	Object Resolver Details	82
3.2.2.1	Abstract Data Model	82
3.2.2.2	Timers	83
3.2.2.3	Initialization	83
3.2.2.4	Higher-Layer Triggered Events	83
3.2.2.5	Message Processing Events and Sequencing Rules	83
3.2.2.5.1	IObjectExporter Methods	83
3.2.2.5.1.1	IObjectExporter::ResolveOxid (Opnum 0)	84
3.2.2.5.1.2	IObjectExporter::SimplePing (Opnum 1)	85
3.2.2.5.1.3	IObjectExporter::ComplexPing (Opnum 2)	86
3.2.2.5.1.4	IObjectExporter::ServerAlive (Opnum 3)	87
3.2.2.5.1.5	IObjectExporter::ResolveOxid2 (Opnum 4)	88
3.2.2.5.1.6	IObjectExporter::ServerAlive2 (Opnum 5)	88
3.2.2.5.1.7	Allocating and Deleting OID Entries	89
3.2.2.5.1.8	Allocating OXID Entries	89
3.2.2.5.2	IActivation and IRemoteSCMAActivator Methods	90
3.2.2.5.2.1	IActivation Methods	90
3.2.2.5.2.2	IRemoteSCMAActivator Methods	90
3.2.2.5.2.2.1	IRemoteSCMAActivator:: DummyQueryInterfaceIRemoteSCMAActivator (Opnum 0)	91

3.2.2.5.2.2.2	IRemoteSCMAActivator:: DummyAddRefIRemoteSCMAActivator (Opnum 1)	91
3.2.2.5.2.2.3	IRemoteSCMAActivator:: DummyReleaseIRemoteSCMAActivator (Opnum 2)	91
3.2.2.5.2.3	IActivation::RemoteActivation, IRemoteSCMAActivator::RemoteGetClassObject, IRemoteSCMAActivator::RemoteCreateInstance	91
3.2.2.5.2.3.1	IActivation:: RemoteActivation (Opnum 0)	92
3.2.2.5.2.3.2	IRemoteSCMAActivator:: RemoteGetClassObject (Opnum 3)	93
3.2.2.5.2.3.3	IRemoteSCMAActivator::RemoteCreateInstance (Opnum 4)	95
3.2.2.6	Timer Events	96
3.2.2.7	Other Local Events	96
4	Protocol Examples	97
4.1	Object Activation + ORPC Call + Release Sequence	97
4.2	QueryInterface + ORPC Call + Release Sequence	98
4.3	Pinging Sequence	99
4.4	OXID Resolution Sequence	100
4.5	IDL Correlation Example for iid_is	101
5	Security	102
5.1	Security Considerations for Implementers	102
5.2	Index of Security Parameters	102
6	Appendix A: Full DCOM IDL.....	103
7	Appendix B: Windows Behavior	112
8	Index.....	120

1 Introduction

The Distributed Component Object Model (DCOM) Remote Protocol is a protocol for exposing application **objects** by way of **remote procedure calls (RPCs)**. The protocol consists of a set of extensions layered on [Microsoft Remote Procedure Call Protocol Extensions](#) as specified in [MS-RPCE].

Note The DCOM Remote Protocol is also referred to as **Object RPC** or ORPC.

1.1 Glossary

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

Activation
Authentication Level
Causality Identifier (CID)
Class Factory
Client
Client Context
CLSID
Component Object Model (COM)
Context
Context Identifier
Context Property
Context Property Identifier
Correlation
Dynamic Endpoint
Endpoint
Envoy Context
Fully Qualified Domain Name (FQDN)
Garbage Collection
Globally Unique Identifier (GUID)
Impersonation Level
Interface
Interface Definition Language (IDL)
Interface Identifier (IID)
Interface Pointer Identifier (IPID)
Little-Endian
Microsoft Interface Definition Language (MIDL)
NetBIOS Name
Network Data Representation (NDR)
Object
Object Class
Object Exporter
Object Exporter Identifier (OXID)
Object Identifier (OID)
Object Reference
Object Resolver
Object RPC (ORPC)
Object Server
OBJREF
Opnum
ORPC Extension
OXID Resolution
Ping Set

Ping Set Identifier (SETID)
Pinging
Protocol Sequence Identifier
Prototype Context
Reference Count
Remote Procedure Call (RPC)
Remote Server Name
Remote Unknown
RPC Protocol Sequence
RPC Transport
Security Provider
Service Principal Name (SPN)
Unicode
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

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

1.2 References

1.2.1 Normative References

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

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

[MS-DTYP] Microsoft Corporation, "[Windows Data Types](#)", March 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](#)", July 2006.

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

1.2.2 Informative References

[MS-DMRP] Microsoft Corporation, "[Disk Management Remote Protocol Specification](#)", July 2006.

[MS-VDS] Microsoft Corporation, "[Virtual Disk Service \(VDS\) Protocol Specification](#)", July 2006.

[MS-WCCE] Microsoft Corporation, "[Windows Client Certificate Enrollment Protocol Specification](#)", July 2006.

[MS-WMI] Microsoft Corporation, "[Windows Management Instrumentation Remote Protocol Specification](#)", July 2006.

[MSDN-CI] Microsoft Corporation, "Client Impersonation", <http://msdn2.microsoft.com/en-us/library/aa376391.aspx>

[MSDN-CLSCTX] Microsoft Corporation, "CLSCTX", <http://msdn2.microsoft.com/en-us/library/ms693716.aspx>

[MSDN-CoGetInstanceFromFile] Microsoft Corporation, "CoGetInstanceFromFile", <http://msdn2.microsoft.com/en-us/library/ms694473.aspx>

[MSDN-CoGetInstanceFromIStorage] Microsoft Corporation, "CoGetInstanceFromIStorage", <http://msdn2.microsoft.com/en-us/library/ms686574.aspx>

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

[MSDN-CoMarshalInterface] Microsoft Corporation, "CoMarshalInterface", <http://msdn2.microsoft.com/en-us/library/ms678428.aspx>

[MSDN-EOLE_AUTHENTICATION_CAPABILITIES] Microsoft Corporation, "EOLE_AUTHENTICATION_CAPABILITIES", <http://msdn2.microsoft.com/en-us/library/ms693368.aspx>

[MSDN-IERRORINFO] Microsoft Corporation, "Component Automation IErrorInfo Interface", <http://msdn2.microsoft.com/en-us/library/ms221233.aspx>

[MSDN-IMarshal] Microsoft Corporation, "IMarshal", <http://msdn2.microsoft.com/en-us/library/ms688712.aspx>

[MSDN-IMessageFilter] Microsoft Corporation, "IMessageFilter", <http://msdn2.microsoft.com/en-us/library/ms693740.aspx>

[MSDN-IPersistFile] Microsoft Corporation, "IPersistFile", <http://msdn2.microsoft.com/en-us/library/ms687223.aspx>

[MSDN-MSHCTX] Microsoft Corporation, "MSHCTX", <http://msdn2.microsoft.com/en-us/library/ms693446.aspx>

[MSDN-MSHLFLAGS] Microsoft Corporation, "MSHLFLAGS", <http://msdn2.microsoft.com/en-us/library/ms680759.aspx>

[MSDN-SS] Microsoft Corporation, "Structured Storage", <http://msdn2.microsoft.com/en-us/library/aa380369.aspx>

[MSDN-STGMC] Microsoft Corporation, "STGM Constants", March 2003, <http://msdn2.microsoft.com/en-us/library/aa380337.aspx>

[MSDN-WSI] Microsoft Corporation, "WTS_SESSION_INFO", <http://msdn2.microsoft.com/en-us/library/aa383864.aspx>

1.3 Protocol Overview (Synopsis)

The Distributed Component Object Model (DCOM) Remote Protocol extends the **Component Object Model (COM)** over a network by providing facilities for creating and activating objects, and for managing **object references**, object lifetimes, and object **interface** queries. The DCOM Remote

Protocol is built on top of [Remote Procedure Call Protocol Extensions](#), as specified in [MS-RPCE], and relies on its authentication, authorization, and message integrity capabilities. The following diagram shows the layering of the protocol stack.

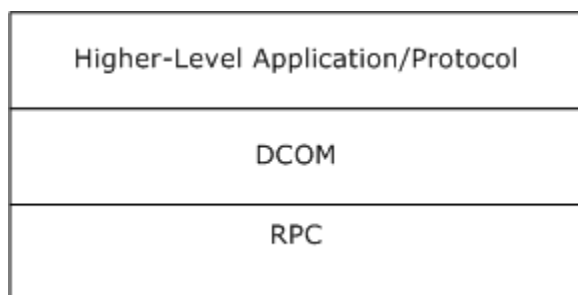


Figure 1: Distributed component object model protocol stack

The following diagram presents an overview of the protocol.

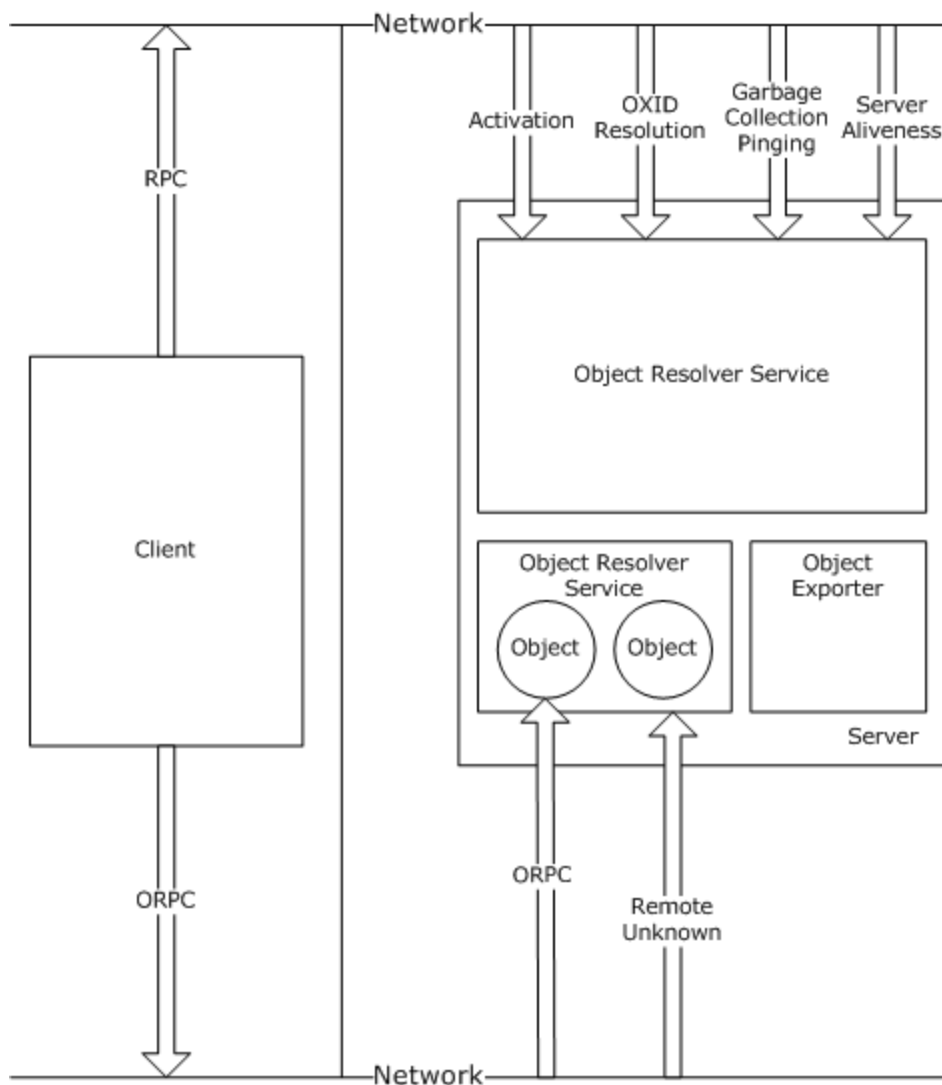


Figure 2: Distributed component object model protocol overview

Higher-level applications use the DCOM **client** to obtain object references and make ORPC calls on the object. The DCOM client in turn uses the Remote Procedure Call Protocol Extensions, as specified in [MS-RPCE], to communicate with the **object server**.

The object server constitutes an **object resolver** service and one or more **object exporters**. Objects are contained in object exporters. Objects are the target of the ORPC calls from the client.

1.3.1 Activation

Activation is a generic term used to describe the act of creating (or sometimes finding) an existing DCOM object or **class factory**. There are two RPC interfaces in the DCOM Remote Protocol that are used to activate objects: [IActivation](#) methods and [IRemoteSCMAActivator](#) methods. At a rudimentary level, activation consists of sending the following to the object activation service on the remote machine:

- A **CLSID**.
- One or more **IIDs**.
- Optionally, an initialization storage reference.

The CLSID identifies the class of the object to be created. The IIDs identify the interfaces on the newly created object that the client is asking for, and if specified, the storage reference identifies some persistent store with which the newly created object is to be initialized after creation.

Activation returns object references to the client application. The client application may also send or receive object references as part of ORPC calls.

1.3.2 Object References

Object references are marshaled as **OBJREF** types. When an OBJREF type is marshaled in the DCOM Remote Protocol, **Network Data Representation (NDR)** instructs the DCOM runtime to write out an OBJREF wrapped inside an [MInterfacePointer](#) into the request/response PDU stream. The marshaled data contains the information required by the client to create the RPC binding back to the object. Similarly, when an OBJREF type is unmarshaled in the DCOM Remote Protocol, NDR instructs the DCOM runtime to construct the object reference using the marshaled data contained in the stream. The DCOM Remote Protocol returns the object reference to the application.

1.3.3 Object Exporter

An object exporter is a conceptual container where objects are created, called, and released. An object must be contained within a single object exporter and must not span multiple object exporters. The protocol is intentionally vague about what an object exporter actually entails. An object exporter may be a thread, a process, or a machine. Clients should not assume implementation details about object exporters. For example, if two objects belong to the same object exporter, clients should not assume that both of the objects reside in the same thread, process, or machine.

An object exporter listens on the network by way of RPC protocols.

An object exporter contains a **Remote Unknown** object, which supports the following ORPC interfaces:

[IRemUnknown](#) interface: An ORPC interface that contains methods used to call QueryInterface, AddRef, and Release on remote objects.

[IRemUnknown2](#) interface: An ORPC interface that extends the functionality of IRemUnknown.

The client uses the AddRef and Release methods to manage the lifetime of objects contained in the object exporter. The client uses the QueryInterface method to obtain object references for additional interface types implemented by an object.

An object exporter is identified by its **OXID**. When a client receives an OXID as part of an object reference, it must determine the RPC binding information required to communicate with the Remote Unknown object of the object exporter. The client uses the **OXID resolution** (see section [3.1.4.1.2.2](#)) mechanism to achieve this.

1.3.4 ORPC Calls

An ORPC call is equivalent to, and possesses a one-to-one correspondence with, RPC calls. ORPC calls are distinguished from RPC calls by the contents of the Object **UUID** field of the RPC header, as

specified in [C706] section 12.5.2.6. In the DCOM Remote Protocol, the Object UUID field carries an **interface pointer identifier (IPID)** specifying the interface targeted by a given ORPC call on an object.

ORPC calls are further distinguished from RPC calls in that the former will always have implicit additional parameters present within the request and response buffers for each call. These additional parameters are referred to as ORPCTHIS and ORPCTHAT respectively (see section 2.2.1.12). The ORPCTHIS and ORPCTHAT parameters are conceptually and syntactically placed ahead of all other values in the RPC PDU body (as specified in [C706] section 12.1).

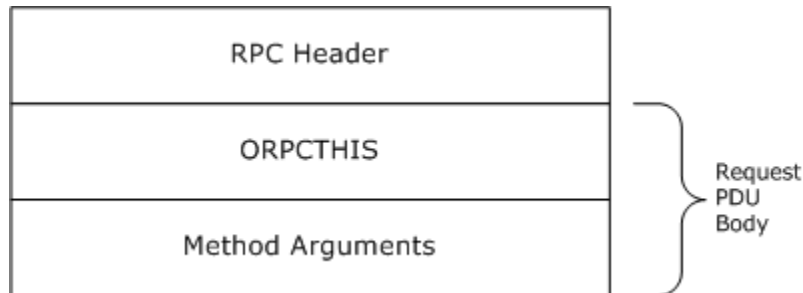


Figure 3: Object RPC calls and the PDU body request

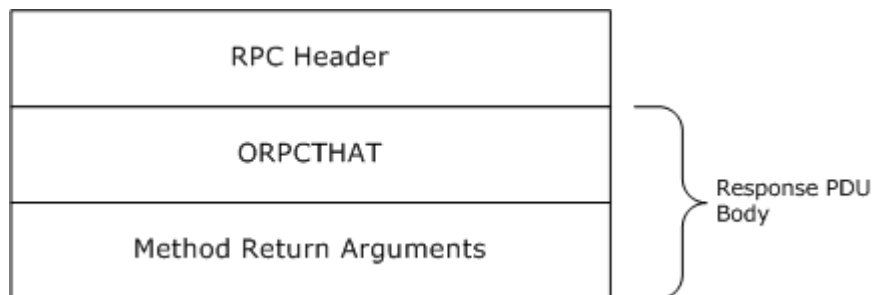


Figure 4: Object RPC calls and the PDU body response

The ORPCTHIS and ORPCTHAT arguments are used to provide versioning, causality information, and the capability to send application-specific out-of-band data.

1.3.5 Causality Identifiers

Each ORPC call carries with it, within the **ORPCTHIS** structure, a **GUID** known as the **causality identifier (CID)**. The CID connects a chain of ORPC calls that are causally related. Object exporters MAY use the CID to provide synchronization around ORPC calls. They may also use the CID to prevent deadlocks within ORPC calls.

If a new ORPC call is made from a client that is already executing an ORPC call, the new call must be assigned the same CID as the existing call. If a new ORPC call is made from a client that is not already executing an ORPC call, then a new CID must be allocated for it. For more information, see section 3.1.4.2.

An object exporter must use the CID of an incoming ORPC call to detect whether it belongs to the same causality chain as that of a currently executing outgoing ORPC. If the incoming and outgoing CIDs are not the same, the object exporter MAY not process the incoming ORPC until the outgoing ORPC completes. However, if they are the same, the object exporter MUST process the incoming ORPC; otherwise, a deadlock occurs. For details, see section 3.2.1.5.4.

1.3.6 Reference Counts

The DCOM Remote Protocol uses **reference counts** to manage object lifetimes. Each interface on an object has an associated reference count that governs its lifetime. There are two types of reference counts associated with an interface: public references and private references. The sole distinction between public and private references is that private references may only be released by the client identity that requested them.

To ensure that object resources are recovered in the event of machine failures or network failures, the DCOM Remote Protocol incorporates a **garbage collection** mechanism. The mechanism is based on keep-alive **pinging**, which allows a client to maintain the lifetimes of its object references. If an object server fails to receive pings for an object, then eventually the object server reclaims the object. For details, see sections [3.1.6.1](#), [3.2.1.6.2](#), and [3.2.2.6](#).

1.3.7 Object Resolver Service

The object resolver service is the part of the DCOM Remote Protocol that performs activation (see section [3.1.4.1.1](#)), OXID resolution (see section [3.2.2.5.1.1](#)), garbage collection (see sections [3.2.1.6.2](#) and [3.2.2.6](#)), and server aliveness tests (see section [3.2.2.5.1.6](#)). The object resolver service can be reached as specified in sections [1.9](#) and [3.2.2.3](#). The object resolver service implements the following RPC interfaces:

[IObjectExporter](#) methods.

[IActivation](#): contains a method used to create objects and class factories.

[IRemoteSCMAActivator](#): contains more methods used to create objects and class factories.

1.4 Relationship to Other Protocols

The DCOM Remote Protocol is built on top of [Remote Procedure Call Protocol Extensions](#), as specified in MS-RPCE. As described in section [2.1](#), the DCOM Remote Protocol uses additional buffer space at the beginning of the RPC PDU body for passing out-of-band data that is not part of the method call signature.

The following protocols are layered above the DCOM Remote Protocol:

- Windows Client Certificate Enrollment Protocol (as specified in [\[MS-WCCE\]](#)).
-
- Disk Management Remote Protocol (as specified in [\[MS-DMRP\]](#)).
- Virtual Disk Service (VDS) Protocol (as specified in [\[MS-VDS\]](#)).
- Windows Management Instrumentation Remote Protocol (as specified in [\[MS-WMI\]](#)).

1.5 Prerequisites/Preconditions

The DCOM Remote Protocol requires that both client and object servers possess implementations of [Remote Procedure Call Protocol Extensions](#), as specified in MS-RPCE. In addition, on the server, the object resolver must be running and reachable, as specified in section [3.2.2.3](#).

1.6 Applicability Statement

The DCOM Remote Protocol is useful and appropriate when a distributed object-based architecture is required. The DCOM Remote Protocol is supported on Windows-based platforms starting with Windows NT.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- Supported Transports: The DCOM Remote Protocol MUST be implemented on top of at least one of the **RPC transports** described in section [2.1](#).
- Protocol Versions: The DCOM Remote Protocol MUST use an RPC version of 0.0 for all RPC and ORPC interfaces. At the DCOM Remote Protocol level, a major and minor version numbering scheme is maintained (see section [2.2.1.11](#)). The major version MUST be 5. The minor version MUST be one of the following: 1, 2, 3, 4, 6, 7. A minor version of 5 is unused and MUST NOT appear in any capability negotiation. The minor versions signify the addition of various capabilities to the protocol. For example, minor version 2 signifies the addition of the ResolveOXID2 method to the IObjectExporter interface (see section [3.2.2.5.1.5](#)).
- Security and Authentication Methods: The DCOM Remote Protocol SHOULD use the underlying security and authentication services provided by RPC.
- Capability Negotiation: The protocol MUST perform explicit capability negotiation, as specified in this section.

The DCOM Remote Protocol implements version negotiation through the following two mechanisms:

1. By the availability of an RPC method or interface on the server; the unavailability of that method or interface implies a certain version to the client, which then undertakes a fallback action as appropriate.
2. By use of the **COMVERSION** structure, which is passed between client and server, clients and servers associate specific version numbers with specific capabilities and behaviors.

The first mechanism is used at the initiation of the protocol, when the client has no knowledge of the capabilities of the server. The second mechanism is used within the operation of the protocol when the **COMVERSION** can be sent or received.

A client detects the version of a server using one of the following mechanisms:

1. By calling either the [IObjectExporter::ServerAlive2 \(Opnum 5\)](#) method or the **IObjectExporter::ResolveOxid2** method on the object resolver. If the server does not support either of these methods, the client assumes that the server supports COM version 5.1. Otherwise, the server returns its version explicitly as a return argument during the method call.
2. During an activation, a server returns its version to the client either as a return argument from the [IActivation::RemoteActivation \(Opnum 0\)](#) method, or as a field of the customREMOTE_REPLY_SCM_INFO structure contained in the [ScmReplyInfoData](#) property returned by either [IRemoteSCMActivator::RemoteGetClassObject \(Opnum 3\)](#) or [IRemoteSCMActivator::RemoteCreateInstance \(Opnum 4\)](#).

Clients MUST NOT call servers with non-matching major versions. Clients MUST compute the lower of the client and the server minor versions and MUST pass this computed version as the client minor version when making activation or ORPC calls. For example, if the client minor version is 7 and the

server minor version is 4, the client MUST specify 4 as its minor version when making activation or ORPC calls.

Servers MUST reject activation requests or ORPC calls from clients with non-matching major versions or higher minor versions.

For more information on the capabilities introduced in each **DCOM** version, see [2.2.1.11](#)

1.8 Vendor-Extensible Fields

The DCOM Remote Protocol uses HRESULTs, which are vendor-extensible. Vendors are free to choose their own values for this field, as long as the C bit (0x20000000) is set, indicating that it is a customer code, as specified in [\[MS-ERREF\]](#) section 2.1.

The **error_status_t** return values used by this protocol are Win32 error codes as specified in [\[MS-ERREF\]](#) section 2.2. Vendors SHOULD reuse those values with their indicated meanings. Choosing any other value runs the risk of a collision in the future.[<1>](#)

1.9 Standards Assignments

The DCOM Remote Protocol object resolver service either MUST use the same **well-known endpoints** as the RPC **endpoint** mapper (as specified in [\[MS-RPCE\]](#) section 2.1), or it MUST register its interfaces with the RPC endpoint mapper service.[<2>](#)

The following is a table of well-known GUIDs in the DCOM Remote Protocol.

Name	GUID	Purpose	Section
CLSID_ActivationContextInfo	{000001a5-0000-0000-c000-000000000046}	Activation property CLSID for ActivationContextInfoData	2.2.1.21.2.5
CLSID_ActivationPropertiesIn	{00000338-0000-0000-c000-000000000046}	OBJREF_CUSTOM unmarshaller CLSID for ActivationPropertiesIn	3.2.2.5.2.3.2 3.2.2.5.2.3.3
CLSID_ActivationPropertiesOut	{00000339-0000-0000-c000-000000000046}	OBJREF_CUSTOM unmarshaller CLSID for ActivationPropertiesOut	3.2.2.5.2.3.2 3.2.2.5.2.3.3
CLSID_CONTEXT_EXTENSION	{00000334-0000-0000-c000-000000000046}	ORPC_EXTENT identifier for Context ORPC extension	2.2.1.20.4
CLSID_ContextMarshaler	{0000033b-0000-0000-c000-000000000046}	OBJREF_CUSTOM unmarshaller CLSID for contexts	2.2.1.19
CLSID_ERROR_EXTENSION	{0000031c-0000-0000-c000-000000000046}	ORPC_EXTENT identifier for Error information ORPC extension	2.2.1.20.1
CLSID_ErrorObject	{0000031b-0000-0000-c000-000000000046}	OBJREF_CUSTOM unmarshaller CLSID for error information	2.2.1.20.2
CLSID_InstanceInfo	{000001ad-0000-0000-c000-000000000046}	Activation property CLSID for InstanceInfoData	2.2.1.21.2.3

Name	GUID	Purpose	Section
	000000000046}		
CLSID_InstantiationInfo	{000001ab-0000-0000-c000-000000000046}	Activation property CLSID for InstantiationInfoData	2.2.1.21.2.1
CLSID_PropsOutInfo	{00000339-0000-0000-c000-000000000046}	Activation property CLSID for PropsOutInfo	2.2.1.21.2.9
CLSID_ScmReplyInfo	{000001b6-0000-0000-c000-000000000046}	Activation property CLSID for ScmReplyInfoData	2.2.1.21.2.8
CLSID_ScmRequestInfo	{000001aa-0000-0000-c000-000000000046}	Activation property CLSID for ScmRequestInfoData	2.2.1.21.2.4
CLSID_SecurityInfo	{000001a6-0000-0000-c000-000000000046}	Activation property CLSID for SecurityInfoData	2.2.1.21.2.7
CLSID_ServerLocationInfo	{000001a4-0000-0000-c000-000000000046}	Activation property CLSID for LocationInfoData	2.2.1.21.2.6
CLSID_SpecialSystemProperties	{000001b9-0000-0000-c000-000000000046}	Activation property CLSID for SpecialPropertiesData	2.2.1.21.2.2
IID_IActivation	{4d9f4ab8-7d1c-11cf-861e-0020af6e7c57}	RPC interface UUID for IActivation	3.2.2.5.2.1
IID_IActivationPropertiesIn	{000001A2-0000-0000-C000-000000000046}	The value of the iid field of the <i>pActProperties</i> OBJREF structure	3.2.2.5.2.3.2 3.2.2.5.2.3.3
IID_IActivationPropertiesOut	{000001A3-0000-0000-C000-000000000046}	The value of the iid field of the <i>ppActProperties</i> OBJREF structure	3.2.2.5.2.3.2 3.2.2.5.2.3.3
IID_IContext	{000001c0-0000-0000-C000-000000000046}	The value of the iid field of the Context structure.	2.2.1.19
IID_IObjectExporter	{99fcfec4-5260-101b-bbcb-00aa0021347a}	RPC interface UUID for IObjectExporter	3.2.2.5.1
IID_IRemoteSCMAActivator	{000001A0-0000-0000-C000-000000000046}	RPC interface UUID for IRemoteSCMAActivator	3.2.2.5.2.2
IID_IRemUnknown	{00000131-0000-0000-C000-000000000046}	RPC interface UUID for IRemUnknown	3.2.1.5.6

Name	GUID	Purpose	Section
IID_IRemUnknown2	{00000143-0000-0000-C000-000000000046}	RPC interface UUID for IRemUnknown2	3.2.1.5.7.1
IID_IUnknown	{00000000-0000-0000-C000-000000000046}	RPC interface UUID for IUnknown	3.2.1.5.8

2 Messages

The following sections specify how DCOM Remote Protocol messages are transported and details of message syntax, including common data types, extensions, and activation properties.

2.1 Transport

DCOM is based on RPC, and implementations SHOULD support the use of any **RPC protocol sequence** available in the underlying RPC implementation. The client SHOULD discover an initial working RPC protocol by calling the object resolver on multiple protocols. [IObjectExporter::ServerAlive2 \(Opnum 5\)](#) SHOULD be used for this purpose, and then any RPC protocol to which the object resolver responds SHOULD be used.

The object resolver and any given object exporter MUST indicate their supported RPC protocols through an array of STRINGBINDING structures contained in the [DUALSTRINGARRAY](#) structure. The DUALSTRINGARRAY structure is returned from the server to the client through various methods in the protocol.

The object resolver service MUST be reachable at either well-known endpoints or through the RPC endpoint mapper, as specified in section [1.9](#).

Object resolvers and object exporters MUST always support the OXID resolution mechanism specified in section [3.1.4.1.2.2](#), even if the object exporters use well-known endpoints. Object resolvers and object exporters MUST NOT rely on clients to know the endpoint other than obtaining it through the protocol.

The interface version of all object (ORPC) interfaces MUST be 0.0. DCOM does not support versioning of ORPC interfaces through the RPC interface versioning mechanism. Instead, all DCOM interfaces MUST be immutable once defined.

The interface version of all native (non-object) RPC interfaces used in the protocol MUST be 0.0.

ORPC calls are equivalent to, and possess a one-to-one correspondence with, RPC calls. ORPC calls are distinguished from RPC calls by the contents of the Object UUID field of the RPC header, as specified in [\[C706\]](#) section [12.5.2.6](#). In DCOM, the Object UUID field carries an IPID specifying the interface targeted by a given ORPC call on an object.

ORPC calls are further distinguished from RPC calls in that the former will always have implicit additional parameters present within the request and response buffers for each call. These additional parameters are referred to as **ORPCTHIS** and **ORPCTHAT** respectively; see section [2.2.1.12](#). The **ORPCTHIS** and **ORPCTHAT** values are conceptually and syntactically placed ahead of all other values in the RPC PDU body (as specified in [\[C706\]](#) section [12.1](#)).

RPC PDU types other than Request and Response PDUs (as specified in [\[C706\]](#) section [12](#)) are used by DCOM without extension or modification, and therefore are not discussed further in this specification.

DCOM uses security, as described in [\[MS-RPCE\]](#) section 1.7.

The server SHOULD register one or more security providers specified in [\[MS-RPCE\]](#) section 2.2.1.1.7; the choice of security provider is implementation-dependent. [<3>](#)

2.2 Message Syntax

All structures are defined in the **IDL** syntax and are marshaled as specified in [\[C706\]](#) part [3](#). The IDL is documented in section [6](#).

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

All extra padding bytes **MUST** be zero, unless otherwise specified, and **MUST** be ignored upon receipt.

Note that LPWSTR or WCHAR* types specified in an IDL structure that are annotated with the [string] attribute **MUST** be null-terminated, as specified in [\[C706\]](#) section [14.3.4](#).

2.2.1 Common Data Types

The following sections contain common data types.

2.2.1.1 OID

The **OID** type specifies an **Object Identifier (OID)**.

This type is declared as follows:

```
typedef unsigned hyper OID;
```

2.2.1.2 SETID

The **SETID** type specifies a **Ping Set Identifier (SETID)**.

This type is declared as follows:

```
typedef unsigned hyper SETID;
```

2.2.1.3 HRESULT

This specification uses the HRESULT type, as specified in [\[MS-DTYP\]](#) section 2.2.17.

2.2.1.4 error_status_t

This specification uses the error_status_t type, as specified in [\[C706\]](#) section [4.2.26](#) and in section [1.8](#) of this protocol specification.

2.2.1.5 GUID

The GUID type specifies a globally unique identifier (GUID), as specified in [\[MS-DTYP\]](#), section [2.3.2](#).

2.2.1.6 CID

The **CID** type specifies a causality identifier (CID).

This type is declared as follows:

```
typedef GUID CID;
```

2.2.1.7 CLSID

The **CLSID** type specifies a class identifier (CLSID) for a GUID that identifies an **object class**.

This type is declared as follows:

```
typedef GUID CLSID;
```

2.2.1.8 IID

The **IID** type specifies an interface identifier (IID).

This type is declared as follows:

```
typedef GUID IID;
```

2.2.1.9 IPID

The **IPID** type specifies an interface pointer identifier (IPID).

This type is declared as follows:

```
typedef GUID IPID;
```

2.2.1.10 OXID

The **OXID** type specifies an object exporter identifier (OXID).

This type is declared as follows:

```
typedef hyper OXID;
```

2.2.1.11 COMVERSION

The **COMVERSION** structure is used to specify the major and minor version of either the client or the server DCOM Remote Protocol implementation.

```
typedef struct tagCOMVERSION {  
    unsigned short MajorVersion;  
    unsigned short MinorVersion;  
} COMVERSION;
```

MajorVersion: This MUST contain the major version of the DCOM Remote Protocol.

MinorVersion: This MUST contain the minor version of the DCOM Remote Protocol.

The following table specifies the capabilities introduced in each DCOM version. [<4>](#)

Version	Change
5.1	Initial DCOM Remote Protocol release.
5.2	Added ResolveOxid2 to the IObjectExporter interface; see section 3.2.2.5.1.5 .
5.3	MUST NOT be used.
5.4	Update in the marshaling of arrays of interface pointers. Update in the marshaling of conformant embedded structures.
5.5	Unused. This is to avoid having a DCOM version with matching major and minor version numbers.
5.6	Added OBJREF_HANDLER and OBJREF_EXTENDED to the OBJREF type. Added IRemoteSCMAActivator interface methods (see section 3.2.2.5.2.2). Added IObjectExporter::ServerAlive2 (Opnum 5) method to IObjectExporter interface. Added IRemUnknown2 interface.
5.7	No DCOM changes from 5.6. <5>

2.2.1.12 object IDL Attribute

The object IDL Attribute is an IDL extension to the interface attributes specified in [\[C706\]](#) section [4.2.4](#).

This IDL extension specifies that calls on methods in the attributed interface MUST be ORPC calls.

2.2.1.13 ORPCTHIS and ORPCTHAT

2.2.1.13.1 ORPC_EXTENT

ORPC_EXTENT is a binary large object (BLOB) of data whose format is identified by a GUID. It is used on DCOM Remote Protocol calls to pass arbitrary out-of-band data that is not part of the explicit method signature. Unless otherwise specified, clients and servers MUST ignore ORPC_EXTENTs whose format they do not recognize. [<6>](#)

```
typedef struct tagORPC_EXTENT {
    GUID id;
    unsigned long size;
    [size_is((size+7)&~7)] byte data[];
} ORPC_EXTENT;
```

id: This MUST contain a GUID that identifies the format of the opaque data in the data field.

size: This MUST specify the size, in bytes, in the data field excluding any padding bytes that were added to round the array size to a multiple of 8.

data: This MUST contain an array of bytes that form the extent data. The array size MUST be a multiple of 8 for alignment reasons.

2.2.1.13.2 ORPC_EXTENT_ARRAY

ORPC_EXTENT_ARRAY is an array of [ORPC_EXTENT](#) structures.

```
typedef struct tagORPC_EXTENT_ARRAY {
    unsigned long size;
    unsigned long reserved;
    [size_is((size+1)&~1,), unique]
    ORPC_EXTENT** extent;
} ORPC_EXTENT_ARRAY;
```

size: This MUST specify the number of non-NULL elements in the extent field.

reserved: This MUST be set to zero, and MUST be ignored on receipt.

extent: This MUST be an array of ORPC_EXTENTs. The array size MUST be a multiple of 2 for alignment reasons.

2.2.1.13.3 ORPCTHIS

The **ORPCTHIS** structure is the first (implicit) argument sent in an ORPC request PDU, and is used to send **ORPC extension** data to the server. The **ORPCTHIS** structure is also sent as an explicit argument in activation RPC requests.

```
typedef struct tagORPCTHIS {
    COMVERSION version;
    unsigned long flags;
    unsigned long reserved1;
```

```

    CID cid;
    [unique] ORPC_EXTENT_ARRAY* extensions;
} ORPCTHIS;

```

version: A [CONVERSION](#) structure that MUST contain the version number of the client. For details, see section [2.2.1.5](#).

flags: This MUST be set to zero, and MUST be ignored on receipt.

reserved1: This MUST be set to zero, and MUST be ignored on receipt.

cid: This MUST contain a causality identifier (CID) for the ORPC call. For details, see section [1.3.5](#).

extensions: If non-NULL, this MUST be a pointer to an [ORPC_EXTENT_ARRAY](#) structure.

2.2.1.13.4 ORPCTHAT

The **ORPCTHAT** structure is the first (implicit) argument returned in an ORPC response PDU, and is used to return ORPC extension data to the client. The **ORPCTHAT** structure is also returned as an explicit argument from an activation request.

```

typedef struct tagORPCTHAT {
    unsigned long flags;
    [unique] ORPC_EXTENT_ARRAY* extensions;
} ORPCTHAT;

```

flags: This MUST be set to zero, and MUST be ignored on receipt.

extensions: If non-NULL, this field MUST contain an [ORPC_EXTENT_ARRAY](#).

2.2.1.14 MInterfacePointer

MInterfacePointer is an NDR-marshaled structure that MUST contain a hand-marshaled [OBJREF](#).

```

typedef struct tagMInterfacePointer {
    unsigned long ulCntData;
    [size_is(ulCntData)] byte abData[];
} MInterfacePointer;

```

ulCntData: This MUST specify the size, in bytes, of the *abData* parameter.

abData: An array of bytes that MUST contain an OBJREF.

2.2.1.15 PMInterfacePointerInternal

The **PMInterfacePointerInternal** type specifies a pointer to an [MInterfacePointer](#) structure.

This type is declared as follows:


```
typedef [disable_consistency_check] MInterfacePointer* PMInterfacePointerInternal;
```

where **disable_consistency_check** is as specified in [\[MS-RPCE\]](#) section 2.2.4.15

2.2.1.16 iid_is IDL Attribute

iid_is IDL Attribute is an IDL extension that specifies the uuid ([\[C706\]](#) section [4.2.4.1](#)) of an interface type method parameter.

An interface type parameter attributed with iid_is MUST be marshaled as an **MInterfacePointer** containing an [OBJREF](#) instance with its iid field set to the value of uuid.

When unmarshaling an interface type parameter attributed with iid_is, the implementation of target level 5.0 Correlation Validation extensions, as specified in [\[MS-RPCE\]](#), SHOULD validate that the iid specified in the OBJREF is equal to the uuid specified by the iid_is attribute. If different, the implementation MUST indicate that the octet stream is invalid, as specified in [\[MS-RPCE\]](#).

For example:

```
HRESULT CreateInstance([in]REFIID riid,  
                      [out, iid_is(riid)]Iunknown** ppUnknown);
```

In this example, the client MUST pass the uuid of the interface parameter, *ppUnknown*, in the *riid* parameter. When returning a successful result, the server MUST marshal the *ppUnknown* parameter as an **MInterfacePointer** containing an OBJREF instance with its iid field set to the value of riid.

2.2.1.17 OBJREF

OBJREF is the marshaled format for a DCOM Remote Protocol object reference. There are four different formats for an OBJREF, which are specified by different definitions of the **u_objref** field. This section defines the initial header information. The following sections define substructures found in the **u_objref** field.

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

signature: This MUST be set to the value 0x574f454d.

flags: This MUST be set to ONE of the following values:

Value	Meaning
OBJREF_STANDARD 0x00000001	u_objref MUST contain an OBJREF_STANDARD .
OBJREF_HANDLER 0x00000002	u_objref MUST contain an OBJREF_HANDLER .
OBJREF_CUSTOM 0x00000004	u_objref MUST contain an OBJREF_CUSTOM .
OBJREF_EXTENDED 0x00000008	u_objref MUST contain an OBJREF_EXTENDED .

iid: The IID for which this OBJREF was marshaled; this MUST NOT be set to GUID_NULL.

The iid MUST be specified either by the uuid IDL attribute (as specified in [\[C706\]](#) section 4.2.4.1) of the interface type method parameter, or it MUST be specified using the [iid is IDL Attribute](#).

u_objref: A structure specified by the value of **flags** above.

2.2.1.17.1 STDOBJREF

STDOBJREF is a supporting structure for three of the four different [OBJREF](#) formats. It is used to specify standard information about an object reference: number of reference counts, the identifying IPID, OXID, and OID.

The STDOBJREF structure is marshaled by the protocol either as a byte array (for example, when it is marshaled as part of an OBJREF) or in the NDR format (for example, when it is returned as part of a [REMQIRERESULT](#) from [IRemUnknown::RemQueryInterface](#)). The field definitions and meanings are identical in both cases. When the STDOBJREF structure is marshaled as a byte array, all of its fields MUST be marshaled in **little-endian** order.

2.2.1.17.2 STDOBJREF (Packet Version)

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

flags: This MUST be zero or the following value: [<7>](#)

Value	Meaning
0x00000000	The client MUST perform garbage collectionping (see section 3.1.6.1) for this object identifier (OID).
SORF_NOPING 0x00001000	The client MUST NOT perform garbage collection ping (see section 3.1.6.1) for this object identifier (OID).

cPublicRefs: The number of public references on the server object, which MUST be released later. For more information, see section [3.1.4.4.2](#).

oxid: This MUST be an OXID identifying the object exporter that contains the object.

oid: This MUST be an OID identifying the object.

ipid: This MUST be an IPID identifying a specific interface on the object.

2.2.1.17.3 STDOBJREF (IDL Version)

The parameter meanings for this structure are identical to those defined in section [2.2.1.17.2](#).

```
typedef struct tagSTDOBJREF {  
    unsigned long flags;  
    unsigned long cPublicRefs;  
    OXID oxid;  
    OID oid;  
    IPID ipid;  
} STDOBJREF;
```

2.2.1.17.4 OBJREF_STANDARD

This form of OBJREF is the simplest, combining an [STDOBJREF](#) structure with a [DUALSTRINGARRAY](#) structure. It is used when there is no need to utilize the extra fields offered by the other OBJREF formats.

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

std: This MUST be an STDOBJREF.

saResAddr: A DUALSTRINGARRAY that MUST contain the network and security bindings for the object resolver service on the server.

2.2.1.17.5 OBJREF_HANDLER

This form of OBJREF is used by the server object to provide an identifier for a helper object on the client, which the client may use as a proxy for the server object.

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

- std:** This MUST specify an [STDOBJREF](#).
- clsid:** This MUST specify a CLSID identifying an object class on the client that the client uses as a handler for the interface identified by the iid field of the containing OBJREF.
- saResAddr:** This MUST specify a [DUALSTRINGARRAY](#) that MUST contain the network and security bindings for the object resolver service on the server.

2.2.1.17.6 OBJREF_CUSTOM

This form of OBJREF is used by a server object to marshal itself into an opaque binary large object (BLOB) using a custom marshaler. The custom marshaler is a COM object that can marshal and

unmarshal the data contained in the BLOB. The CLSID of the custom marshaler object's object class is specified within the OBJREF.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	1	2	3	4	5	6	7	8	9	30	1
clsid																															
...																															
...																															
...																															
cbExtension																															
size																															
pObjectData (variable)																															
...																															

clsid: This MUST specify a CLSID, supplied by an application or higher-layer protocol, identifying an object class associated with the data in the **pObjectData** field.[<8>](#8)

cbExtension: This MUST be set to zero, and MUST be ignored on receipt.

size: This MUST contain the size, in bytes, of the **pObjectData** member.

pObjectData: This MUST be an array of bytes containing data supplied by an application or higher-layer protocol.

2.2.1.17.7 OBJREF_EXTENDED

The OBJREF_EXTENDED format is used when the server returns a marshaled **envoy context** to the client.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	1	2	3	4	5	6	7	8	9	30	1
std																															
...																															
...																															

...
...
...
...
...
(std cont'd for 2 rows)
Signature1
saResAddr (variable)
...
nElms
Signature2
ElmArray (variable)
...

std: This MUST contain an [STDOBJREF](#) structure.

Signature1: This MUST be set to 0x4E535956.

saResAddr: This MUST contain a [DUALSTRINGARRAY](#) structure containing network and security bindings for the object resolver service on the server.

nElms: The number of elements in the **ElmArray** field. This field MUST be set to 0x00000001.
(Note that while this protocol supports only a single element, for historical reasons the protocol uses an array of one element.)

Signature2: This MUST be set to 0x4E535956.

ElmArray: This MUST be a [DATAELEMENT](#) entry.

2.2.1.17.8 DATAELEMENT

The DATAELEMENT structure is used to identify and marshal an envoy context as part of a larger [OBJREF_EXTENDED](#) structure.

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

dataID: This MUST specify a **context identifier** for the marshaled context. This MUST NOT be set to GUID_NULL.

cbSize: The unsigned number of bytes present in the **Data** field, excluding any padding bytes at the end of the **Data** field that were added to round the array size to an integral multiple of eight bytes. This MUST NOT be 0.

cbRounded: The unsigned size, in bytes, of the **Data** field. The **cbRounded** value MUST equal the **cbSize** value, rounded up to a multiple of eight.

Data: An array of **cbRounded** bytes that MUST contain a marshaled envoy context; see section [2.2.1.19](#).

2.2.1.18 DUALSTRINGARRAY

The DUALSTRINGARRAY structure contains an array of string bindings and an array of security bindings. The protocol uses this structure as a fundamental means of specifying RPC addressability and security information for either an object resolver or an object exporter.

2.2.1.18.1 DUALSTRINGARRAY (Packet Version)

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
wNumEntries																wSecurityOffset															
StringBinding (variable)																															
...																															
nullterm1																SecBinding (variable)															
...																															
nullterm2																															

wNumEntries: The (unsigned) size in unsigned shorts from the first entry in the **StringBinding** array to the end of the buffer.

wSecurityOffset: The (unsigned) size in unsigned shorts from the first entry in the StringBinding array to the first entry in the **SecBinding** array.

StringBinding: An array of [STRINGBINDING](#) structures that SHOULD be ordered in decreasing order of preference by the object server.

nullterm1: This MUST contain zero to denote the end of the array of STRINGBINDING entries.

SecBinding: An array of [SECURITYBINDING](#) structures that SHOULD be ordered in decreasing order of preference by the object server.

nullterm2: This MUST contain zero to denote the end of the array of SECURITYBINDING entries.

A DUALSTRINGARRAY MUST contain at least one STRINGBINDING structure and one SECURITYBINDING structure, although either or both of these MAY be empty. As a result, the smallest possible DUALSTRINGARRAY structure will consist of wNumEntries=4 and wSecurityOffset=2 followed by two NULL Unicode characters.

2.2.1.18.2 DUALSTRINGARRAY (IDL Version)

In certain cases in the DCOM Remote Protocol, a **DUALSTRINGARRAY** is passed or returned as a parameter in an RPC call. For example, see section [3.2.2.5.2.3.1](#). In all such cases, the IDL definition that the DCOM Remote Protocol uses is as follows:

```
typedef struct tagDUALSTRINGARRAY {
    unsigned short wNumEntries;
    unsigned short wSecurityOffset;
    [size_is(wNumEntries)] unsigned short aStringArray[];
} DUALSTRINGARRAY;
```

wNumEntries: This MUST contain the total number of unsigned shorts in the **aStringArray** field.

wSecurityOffset: This MUST be the offset in number of elements from the beginning of the **aStringArray** array to the beginning of the first security binding within the array. For details, see section [2.2.1.18.1](#).

aStringArray: This MUST be an array of **wNumEntries** unsigned shorts. This field MUST be interpreted to contain a sequence of [STRINGBINDING](#) entries followed by a sequence of [SECURITYBINDING](#) entries, in the same syntax as defined in section [2.2.1.18.1](#).

2.2.1.18.3 STRINGBINDING

The STRINGBINDING structure describes an RPC protocol, a network address, and optionally an RPC endpoint for the RPC protocol that a client may use to communicate with either an object resolver or an object exporter.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
wTowerId																aNetworkAddr (variable)															
...																															

wTowerId: An unsigned short that MUST contain an RPC **protocol sequence identifier** constant that identifies the protocol to be used in RPC calls to the server named in the **aNetworkAddr** field, as specified in [\[C706\]](#) table I-2 [Appendix I](#). Additionally, if this field is zero, the **aNetworkAddr** field MUST NOT be present. If this field is not zero, the **aNetworkAddr** field MUST be present.[<9>](#)

aNetworkAddr: An optional null-terminated Unicode string that, if present, MUST contain a network address and optionally RPC endpoint information. This field MUST use one of the two following formats:

The first format is "servername", where "servername" contains a machine name that conforms to the RPC protocol syntax identified by **wTowerId** (as specified in [\[C706\]](#) section [6.2.1](#)). For example, for the NCACN_IP_TCP RPC protocol sequence, the servername MUST be a **NetBIOS Name**, an IP address or a **fully qualified domain name (FQDN)**. This is the format used when a DUALSTRINGARRAY containing object resolver bindings is returned; for example, returned from [IObjectExporter::ServerAlive2](#) or in an [OBJREF](#). Endpoint information is omitted, as this information is assumed, as specified in [\[MS-RPCE\]](#) section 2.1.

The second format is "servername[endpoint]", for example, "10.0.0.1[5000]", where "servername" and "endpoint" specify a machine name and an RPC endpoint, both of which conform to the RPC protocol syntax identified by **wTowerId**. This is the format used when a [DUALSTRINGARRAY](#) containing object exporter bindings is returned; for example, from [IObjectExporter::ResolveOxid2](#) or [IActivation::RemoteActivation](#). The contents of endpoint is RPC protocol-specific, as specified in [\[C706\]](#) section [6.2.2](#).

An empty STRINGBINDING structure contains a **wTowerId** whose value is zero.

2.2.1.18.4 SECURITYBINDING

The SECURITYBINDING structure describes an RPC authentication service and a **service principal name (SPN)**. A client uses these to communicate with either an object resolver or an object exporter.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
wAuthnSvc																Reserved (optional)															
aPrincName (variable)																															
...																															

wAuthnSvc: An unsigned short that MUST contain an RPC authentication service constant. For details, see section [2.2.1.26.2](#). Additionally, if this field is zero, the **Reserved** and **aPrincName** fields MUST NOT be present, and this indicates that the client MUST NOT use any security on the ORPC invocations (see section [3.1.4.2](#)). If this field is not zero, the **Reserved** and **aPrincName** fields MUST be present.

Reserved: If present, MUST be set to 0xffff on transmission and ignored on receipt.

aPrincName: An optional null-terminated Unicode string that, if present, MUST contain an SPN. An empty null-terminated Unicode string in this field indicates that no SPN is specified, and the client MUST NOT use an SPN for the ORPC invocations (see section [3.1.4.2](#)).

2.2.1.19 Context

This is the marshaled representation of a context. It contains an array of marshaled **context properties**, each of which is represented by a [PROPMARSHALHEADER](#).

There are three types of context structures:

- A **client context**. This type of context is sent in an activation request in the **pIFDClientCtx** field of the **ActivationContextInfoData** structure (section [2.2.1.21.2.5](#)). The context structure MUST be marshaled into an **OBJREF_CUSTOM** structure (section [2.2.1.17.6](#)) whose **clsid** field is set to **CLSID_ContextMarshaler** (section [1.9](#)). The **iid** field of the **OBJREF** portion of the structure MUST be set to a value that an implementation may or may not use as the IID of a local-only interface. [<10>](#)
- A **prototype context**. An application or a higher-layer protocol may instruct a DCOM client to send this type of context in an activation request. The prototype context is sent in the **pIFDPrototypeCtx** field of the **ActivationContextInfoData** structure (section [2.2.1.21.2.5](#)). The context structure MUST be marshaled into an **OBJREF_CUSTOM** structure (section [2.2.1.17.6](#)) whose **clsid** field is set to **CLSID_ContextMarshaler** (section [1.9](#)). The **iid** field of the **OBJREF** portion of the **OBJREF_CUSTOM** structure MUST be set to **IID_IContext** (see section [1.9](#)). The **iid** field of the **OBJREF** portion of the structure MUST be set to a value that an implementation may or may not use as the IID of a local-only interface. [<11>](#)
- An envoy context. An application or a higher-layer protocol may instruct a DCOM server to send this type of context when marshaling an object. The envoy context is sent in the **Data** field of the **ElmArray** field (section [2.2.1.17.8](#)) of an **OBJREF_EXTENDED** structure (section [2.2.1.17.7](#)).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersion																MinVersion															
ContextId																															
...																															
...																															
...																															
Flags																															
Reserved																															
dwNumExtents																															
cbExtents																															
MshlFlags																															
Count																															
Frozen																															
PropMarshalHeader (variable)																															
...																															

MajorVersion: The major version of this context marshaled format. This MUST be set to 0x0001.

MinVersion: The minor version of this context marshaled format. This MUST be set to 0x0001.

ContextId: A GUID identifying the marshaled context.

Flags: This MUST be set to the following value:

Value	Meaning
CTXMSHLFLAGS_BYVAL 0x00000002	The context is marshaled by value (this is the only representation that is valid on the wire).

Reserved: This MUST be set to 0x00000000 and MUST be ignored on receipt.

dwNumExtents: This MUST be set to 0x00000000. [<12>](#)

cbExtents: This MUST be set to 0x00000000. [<13>](#)

MshlFlags: This MUST contain an implementation-specific value that MUST be ignored on receipt. [<14>](#)

Count: The unsigned number of elements in the **PropMarshalHeader** array. This MUST NOT be zero.

Frozen: A Boolean that MUST be set to TRUE (0x00000001) and that MUST be ignored on receipt.

PropMarshalHeader: This MUST be an array of PROPMARSHALHEADER entries.

2.2.1.19.1 PROPMARSHALHEADER

PROPMARSHALHEADER is the marshaled representation of a context property. It contains a **context property identifier** and a context property data buffer supplied by an application or higher-layer protocol. The data buffer contains either an OBJREF that may be of any valid OBJREF format representing a client or prototype context property, or it contains opaque data representing an envoy context property.

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

clsid: This field MUST be either GUID_NULL or it MUST be a CLSID identifying an object class associated with the data in the **ctxProperty** field. If this field is GUID_NULL, **ctxProperty** MUST contain an OBJREF representing the marshaled context property; otherwise, **ctxProperty** MUST contain opaque data representing the marshaled envoy property.

policyId: A GUID that MUST contain a context property identifier for the marshaled context property in ctxProperty.

flags: This MUST be set to ONE of the following values:

Value	Meaning
CPFLAG_PROPAGATE 0x00000001	This context property is part of a prototype context.
CPFLAG_EXPOSE 0x00000002	This context property is part of a client context.
CPFLAG_ENVOY 0x00000004	This context property is part of an envoy context.

cb: A DWORD that MUST contain the size, in bytes, of the **ctxProperty** field.

ctxProperty: A buffer of cb bytes. This buffer MUST contain an OBJREF representing the marshaled context property if **clsid** is GUID_NULL; otherwise, it MUST contain opaque data representing the marshaled envoy property.

2.2.1.20 ORPC Extensions

This section defines the format for well-known ORPC extensions used in the DCOM Remote Protocol. They are well-known as these extensions are defined by the DCOM Remote Protocol itself and can be used by applications or higher-layer protocols without additional specification. These ORPC extensions are sent by applications or higher-layer protocols within the data field of [ORPC_EXTENT](#) structures as part of ORPC calls. Each extension has a well-known identifying GUID that is sent within the id field of the **ORPC_EXTENT**.

2.2.1.20.1 Error Information ORPC Extension

The purpose of this ORPC extension is to provide a means of passing extended error information between clients and servers.

CLSID_ERROR_EXTENSION (see section [1.9](#)) is the identifying GUID for this ORPC extension. This ORPC extension MUST be marshaled into the data field of an [ORPC_EXTENT](#) as an [OBJREF.<15>](#)

2.2.1.20.2 Custom-Marshaled Error Information Format

This section defines the format of an [OBJREF_CUSTOM](#) that, depending on the use of the DCOM Remote Protocol by an application or a higher-layer protocol, MAY be passed as an error information ORPC extension (see section [2.2.1.20.1](#)). CLSID_ErrorObject (see section [1.9](#)) is the unmarshaller CLSID for this OBJREF_CUSTOM. The format of the **pObjectData** field of the OBJREF_CUSTOM for this CLSID is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
dwVersion																															
dwHelpContext																															
iid																															
...																															
...																															
...																															
dwSourceSignature																															
Source (variable)																															
...																															
dwDescriptionSignature																															
Description (variable)																															
...																															
dwHelpFileSignature																															
HelpFile (variable)																															
...																															

dwVersion: A DWORD that MUST be set to zero.

dwHelpContext: A DWORD that MUST contain an implementation-specific value that SHOULD be ignored on receipt.[<16>](#)

iid: An IID that MUST be the IID of the interface returning the error.

dwSourceSignature: A DWORD that MUST be set to 0xFFFFFFFF if the **Source** field is present and that MUST be set to 0 if the **Source** field is not present.

Source: This MUST contain an ErrorInfoString (see [ErrorInfoString \(section 2.2.1.20.3\)](#)) if the **dwSourceSignature** field is set to 0xFFFFFFFF and MUST NOT be present if the **dwSourceSignature** field is set to 0.<17>

dwDescriptionSignature: A DWORD that MUST be set to 0xFFFFFFFF if the **Description** field is present and that MUST be set to 0 if the **Description** field is not present.

Description: This MUST contain an ErrorInfoString (see ErrorInfoString (section 2.2.1.20.3)) if the **dwDescriptionSignature** field is set to 0xFFFFFFFF and MUST NOT be present if the **dwDescriptionSignature** field is set to 0.<18>

dwHelpFileSignature: A DWORD that MUST be set to 0xFFFFFFFF if the **HelpFile** field is present and that MUST be set to 0 if the **HelpFile** field is not present.

HelpFile:

This MUST contain an ErrorInfoString (see ErrorInfoString (section 2.2.1.20.3)) if the **dwHelpFileSignature** field is set to 0xFFFFFFFF and MUST NOT be present if the **dwHelpFileSignature** field is set to 0.<19>

2.2.1.20.3 ErrorInfoString

This packet specifies the format of the string data that is contained in a [Custom-Marshaled Error Info Format \(section 2.2.1.20.2\)](#) packet.

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

dwMax: A DWORD that MUST contain the number of Unicode characters in the Name array including the NULL terminator.

dwOffset: A DWORD that MUST be set to zero.

dwActual: A DWORD that MUST be set to the value of the **dwMax** field.

Name: This MUST contain an implementation-specific NULL-terminated Unicode string and SHOULD be ignored on receipt.

2.2.1.20.4 Context ORPC Extension

CLSID_CONTEXT_EXTENSION is the identifying GUID for this ORPC extension, which is marshaled into the data field of an [ORPC_EXTENT](#) using the structure format below.

A context ORPC extension is used when a client or server passes data associated with a context property along with and in response to an ORPC call.

All fields **MUST** be marshaled with the same endianness as the encompassing RPC PDU.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Signature																															
Version																															
cPolicies																															
cbBuffer																															
cbSize																															
hr																															
hrServer																															
reserved																															
EntryHeader (variable)																															
...																															
PolicyData (variable)																															
...																															

Signature: This **MUST** be set to 0x414E554B.

Version: This **MUST** be set to 0x00010000.

cPolicies: A DWORD that **MUST** contain the number of elements in the EntryHeader array. This **MUST** also be equal to the number of elements in the PolicyData array.

cbBuffer: A DWORD that **MUST** contain an implementation-specific value that **MUST** be ignored on receipt. [<20>](#)

cbSize: A DWORD that **MUST** contain the size (in bytes) from the beginning of this structure to the end of the last EntryHeader.

hr: An [HRESULT](#) that **MUST** be set to zero, and **MUST** be ignored on receipt.

hrServer: An HRESULT that MUST be set to zero when sending from client to server, and MUST be ignored on receipt. When being returned from server to client, this field MAY contain an implementation-specific error value.<21>

reserved: This MUST be set to zero, and MUST be ignored on receipt.

EntryHeader: An array of [EntryHeader](#) structures. The number of elements in the array MUST be specified in cPolicies.

PolicyData: An array of byte arrays. The number of elements in the outer array MUST be specified in cPolicies. The number of elements in the byte array MUST be specified in the **cbEHBuffer** field (see section [2.2.1.20.5](#)) in the corresponding element in the EntryHeader array.

Both the EntryHeader and PolicyData arrays MUST have sizes that are integral multiples of eight, and all padding bytes MUST be initialized to zero and MUST be ignored on receipt.

2.2.1.20.5 EntryHeader

The EntryHeader structure is used to describe an opaque array of bytes associated with a context property within the context ORPC extension.

All fields MUST be marshaled with the same endianness as the encompassing RPC PDU.

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

Signature: A DWORD that MUST be set to 0x494E414E.

cbEHBuffer: A DWORD that MUST contain the size (in bytes) of the opaque policy data buffer corresponding to this EntryHeader. This MUST NOT be zero.

cbSize: A DWORD that MUST contain the size (in bytes) of the offset from the beginning of the context ORPC extension buffer to the beginning of the opaque policy data buffer corresponding to this EntryHeader.

reserved: A DWORD that MUST be set to 0x00000000, and MUST be ignored on receipt.

policyID: A GUID that MUST contain a context property identifier.

2.2.1.21 Activation Properties BLOB

Activation properties constitute a binary large object (BLOB) of data that contains information used to specify the parameters of activation, or to return the results of activation. Activation properties BLOBs are passed as [in] and [out] parameters of the [IRemoteSCMAActivator](#) methods.

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

dwSize: A DWORD that MUST contain the total size (in bytes) from the beginning of the activation properties BLOB to the end of the last entry in the Property array.

dwReserved: This MUST be set to zero, and MUST be ignored on receipt.

CustomHeader: This MUST contain a [CustomHeader](#) marshaled using the NDR Type Serialization 1 engine (as specified in [\[MS-RPCE\]](#)).

Property: This MUST contain an array of structures listed in section [2.2.1.21.2](#) marshaled using the NDR Type Serialization 1 engine (as specified in [\[MS-RPCE\]](#)). The format of the Nth element in the array is identified by the CLSID specified in the Nth entry of the pclsid array in the **CustomHeader** field. The number of elements in the array MUST be specified by the **cIfs** field of the **CustomHeader** and MUST NOT be zero.

2.2.1.21.1 CustomHeader

The **CustomHeader** structure is used to identify the format and ordering of the properties in the activation properties binary large object (BLOB).

```
typedef struct tagCustomHeader {
    DWORD totalSize;
    DWORD headerSize;
```

```

    DWORD cOpaqueData;
    DWORD destCtx;
    [range (MIN_ACTPROP_LIMIT, MAX_ACTPROP_LIMIT)]
        DWORD cIfs;
    CLSID classInfoClsid;
    [size_is(cIfs)] CLSID* pclsid;
    [size_is(cIfs)] DWORD* pSizes;
    [size_is(cOpaqueData)] CustomOpaqueData* opaqueData;
} CustomHeader;

```

totalSize: This MUST be the total size (in bytes) from the beginning of the **CustomHeader** to the end of the last entry in the subsequent Property array of the activation properties BLOB.

headerSize: This MUST be the total size (in bytes) of the **CustomHeader** as marshaled by the NDR Type Serialization 1 engine (as specified in [\[MS-RPCE\]](#)).

cOpaqueData: This MUST be set to zero, and MUST be ignored on receipt.

destCtx: This MUST contain an implementation-specific value that SHOULD be ignored on receipt. [<22>](#)

cIfs: This MUST be the total number of entries in the subsequent Property array of the activation properties BLOB. The value MUST be between MIN_ACTPROP_LIMIT and MAX_ACTPROP_LIMIT (see section [2.2.1.26.3](#)).

classInfoClsid: This MUST be set to GUID_NULL.

pclsid: This MUST specify an array of cIfs CLSIDs; the Nth entry identifies the Nth entry in the Property array of the activation properties BLOB. Each CLSID is used to uniquely identify an activation property. The valid CLSID values are defined in section [1.9](#).

pSizes: This MUST specify an array of cIfs DWORDs, each containing the size (in bytes) of the corresponding property following the **CustomHeader** in the buffer.

opaqueData: This MUST be set to NULL, and MUST be ignored on receipt.

Note that because this field MUST be NULL, the **CustomOpaqueData** structure is never sent or received in the DCOM Remote Protocol. Therefore, the **CustomOpaqueData** structure is not specified in this document.

2.2.1.21.2 Activation Properties

This section documents the list of properties that MAY appear in the Property array of the [Activation Properties BLOB](#) structure.

2.2.1.21.2.1 InstantiationInfoData

The client uses this structure to specify basic details of the object to be activated, including the identifying object CLSID and one or more requested object interfaces.

CLSID_InstatiationInfo (section [1.9](#)) is used to identify this Property in the [CustomHeader.pclsid](#) array.

```

typedef struct tagInstantiationInfoData {

```

```

CLSID classId;
DWORD classCtx;
DWORD actvflags;
long fIsSurrogate;
[range (1, MAX_REQUESTED_INTERFACES)]
    DWORD cIID;
DWORD instFlag;
[size_is(cIID)] IID* pIID;
DWORD thisSize;
COMVERSION clientCOMVersion;
} InstantiationInfoData;

```

classId: This MUST contain the CLSID of the COM object class that the client activates.

classCtx: This MUST contain an implementation-specific value that SHOULD be ignored on receipt. [<23>](#)

actvflags: This MUST contain a bitwise OR of one or more of the following values:

Value	Meaning
ACTVFLAGS_DISABLE_AAA 0x00000002	The object exporter MUST NOT be allowed to execute under the client's identity. The object exporter MUST return E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) if this requirement cannot be met.
ACTVFLAGS_ACTIVATE_32_BIT_SERVE 0x00000004	The object exporter MUST run in a 32-bit address space. The object exporter MUST return REGDB_E_CLASSNOTREG (as specified in [MS-ERREF] section 2.1) if this requirement cannot be met.
ACTVFLAGS_ACTIVATE_64_BIT_SERVE 0x00000008	The object exporter MUST run in a 64-bit address space. The object exporter MUST return REGDB_E_CLASSNOTREG (as specified in [MS-ERREF] section 2.1) if this requirement cannot be met.
ACTVFLAGS_NO_FAILURE_LOG 0x00000020	The server SHOULD NOT log events when a failure occurs during the activation request.

fIsSurrogate: This MUST be set to FALSE (0x00000000) and MUST be ignored on receipt.

cIID: This MUST specify the number of interfaces in the pIID array. This value MUST be between 1 and MAX_REQUESTED_INTERFACES (see section [2.2.1.26.3](#)).

instFlag: This MUST be set to zero and MUST be ignored on receipt.

pIID: This MUST contain an array of IIDs identifying the interfaces that the client requests from the server.

thisSize: This MUST be the size (in bytes) of this structure, as marshaled by the NDR Type Serialization 1 engine (as specified in [\[MS-RPCE\]](#)). SHOULD be ignored on receipt.

clientCOMVersion: This MUST specify the [COMVERSION](#) of the client and MUST be ignored on receipt.

2.2.1.21.2.2 SpecialPropertiesData

The **SpecialPropertiesData** structure contains miscellaneous parameters specified by the client for an activation request.

CLSID_SpecialSystemProperties (see section [1.9](#)) is used to identify this Property in the CustomHeader.pclsid array.

```
typedef struct tagSpecialPropertiesData {
    unsigned long dwSessionId;
    long fRemoteThisSessionId;
    long fClientImpersonating;
    long fPartitionIDPresent;
    DWORD dwDefaultAuthnLvl;
    GUID guidPartition;
    DWORD dwPRTFlags;
    DWORD dwOrigClsctx;
    DWORD dwFlags;
    DWORD dwPid;
    __int64 hwnid;
    DWORD dwReserved[5];
} SpecialPropertiesData;
```

dwSessionId: This MUST contain an implementation-specific value and SHOULD be ignored on receipt. The value 0xFFFFFFFF is reserved across all implementations and MUST be interpreted to mean that the value is not set. [<24>](#)

fRemoteThisSessionId: This MUST be set to TRUE (0x00000001) if dwSessionId is not 0xFFFFFFFF; otherwise this MUST be set to FALSE (0x00000000). This field MUST be ignored on receipt.

fClientImpersonating: This SHOULD be set to FALSE (0x00000000) and MUST be ignored on receipt. [<25>](#)

fPartitionIDPresent: This MUST contain an implementation-specific value and MAY be ignored on receipt. [<26>](#)

dwDefaultAuthnLvl: This MUST contain an implementation-specific value and MUST be ignored on receipt. [<27>](#)

guidPartition: This MUST contain an implementation-specific value specified by higher-layer protocols and MAY be ignored on receipt. [<28>](#)

dwPRTFlags: This MUST be set to zero, and MUST be ignored on receipt.

dwOrigClsctx: This MUST contain an implementation-specific value and SHOULD be ignored on receipt. [<29>](#)

dwFlags: This MUST contain an implementation-specific value and SHOULD [<30>](#) be ignored on receipt.

dwPid: This MUST be set to zero and MUST be ignored on receipt.

hwnid: This MUST be set to zero and MUST be ignored on receipt.

dwReserved: This MUST be filled with zeros and MUST be ignored on receipt.

2.2.1.21.2.3 InstanceInfoData

The **InstanceInfoData** structure contains data related to persistent activations; that is, object activations in which the newly created object is immediately initialized with state from a previously persisted instance of the object. For more information, see [\[MS-COM-MSDN\]](#), [\[MSDN-SS\]](#), and [\[MSDN-IPersistFile\]](#).

CLSID_InstanceInfo (see section [1.9](#)) is used to identify this Property in the CustomHeader.pclsid array.

```
typedef struct tagInstanceInfoData {
    [string] wchar_t* fileName;
    DWORD mode;
    MInterfacePointer* ifdROT;
    MInterfacePointer* ifdStg;
} InstanceInfoData;
```

fileName: This MAY contain a string to be used to initialize the object. [<31>](#)

mode: This MUST contain an implementation-specific value and MAY be ignored on receipt. [<32>](#)

ifdROT: The pointer MUST be set to NULL and MUST be ignored on receipt.

ifdStg: This MAY contain a marshaled OBJREF to be used to initialize the object. [<33>](#)

2.2.1.21.2.4 ScmRequestInfoData

The **ScmRequestInfoData** structure contains a [customREMOTE_REQUEST_SCM_INFO](#) structure.

CLSID_ScmRequestInfo (see section [1.9](#)) is used to identify this Property in the CustomHeader.pclsid array.

```
typedef struct tagScmRequestInfoData {
    DWORD* pdwReserved;
    customREMOTE_REQUEST_SCM_INFO* remoteRequest;
} ScmRequestInfoData;
```

pdwReserved: This MUST be set to NULL, and MUST be ignored on receipt.

remoteRequest: This MUST specify a **customREMOTE_REQUEST_SCM_INFO** structure. This field MUST NOT be NULL.

2.2.1.21.2.4.1 customREMOTE_REQUEST_SCM_INFO

The **customREMOTE_REQUEST_SCM_INFO** structure specifies the protocol sequence identifiers supported by the client.

```
typedef struct _customREMOTE_REQUEST_SCM_INFO {
    DWORD ClientImpLevel;
    [range (0, MAX_REQUESTED_PROTSEQS)]
```

```

    unsigned short cRequestedProtseqs;
    [size_is(cRequestedProtseqs)] unsigned short* pRequestedProtseqs;
} customREMOTE_REQUEST_SCM_INFO;

```

ClientImpLevel: This MUST contain an implementation-specific value that MUST be ignored on receipt. [<34>](#)

cRequestedProtseqs: This MUST contain the number of elements in the **pRequestedProtseqs** array. SHOULD be at least 1.

MAX_REQUESTED_PROTSEQS (see section [2.2.1.26.3](#)) specifies the upper bound for the range of permissible values in cRequestedProtseqs.

pRequestedProtseqs: This MUST contain an array of RPC protocol sequence identifiers supported by the client.

2.2.1.21.2.5 ActivationContextInfoData

The **ActivationContextInfoData** structure passes a client context and optionally a prototype context to the server as part of an activation request.

CLSID_ActivationContextInfo (see section [1.9](#)) is used to identify this Property in the CustomHeader.pclsid array.

```

typedef struct tagActivationContextInfoData {
    long clientOK;
    long bReserved1;
    DWORD dwReserved1;
    DWORD dwReserved2;
    MInterfacePointer* pIFDClientCtx;
    MInterfacePointer* pIFDPrototypeCtx;
} ActivationContextInfoData;

```

clientOK: This MUST be set to FALSE (0x00000000) and MUST be ignored on receipt.

bReserved1: This MUST be set to FALSE (0x00000000) and MUST be ignored on receipt.

dwReserved1: This MUST be set to zero and MUST be ignored on receipt.

dwReserved2: This MUST be set to zero and MUST be ignored on receipt.

pIFDClientCtx: This MUST contain an OBJREF specifying a marshaled client context. The server MUST return RPC_E_INVALID_OBJREF (as specified in [\[MS-ERREF\]](#) section 2.1) if the OBJREF is NULL or invalid.

pIFDPrototypeCtx: This MAY contain an OBJREF specifying a marshaled prototype context, depending on the usage of the DCOM Remote Protocol by a higher-layer protocol. [<35>](#)

2.2.1.21.2.6 LocationInfoData

The **LocationInfoData** structure MUST be present in the [Activation Properties BLOB](#) structure. The server MUST ignore this structure.

CLSID_ServerLocationInfo (see section [1.9](#)) is used to identify this Property in the CustomHeader.pclsid array.

```
typedef struct tagLocationInfoData {
    [string] wchar_t* machineName;
    DWORD processId;
    DWORD apartmentId;
    DWORD contextId;
} LocationInfoData;
```

machineName: This MUST be set to NULL and MUST be ignored on receipt.

processId: This MUST be set to 0 and MUST be ignored on receipt.

apartmentId: This MUST be set to 0 and MUST be ignored on receipt.

contextId: This MUST be set to 0 and MUST be ignored on receipt.

2.2.1.21.2.7 SecurityInfoData

The **SecurityInfoData** structure SHOULD NOT be sent, and MUST be ignored on receipt. [<36>](#)

CLSID_SecurityInfo (see section [1.9](#)) is used to identify this property in the CustomHeader.pclsid array.

```
typedef struct tagSecurityInfoData {
    DWORD dwAuthnFlags;
    COSERVERINFO* pServerInfo;
    DWORD* pdwReserved;
} SecurityInfoData;
```

dwAuthnFlags: This MUST be set to zero and MUST be ignored on receipt.

pServerInfo: This SHOULD be NULL and MUST be ignored on receipt. For details, see section [2.2.1.21.2.7.1.<37>](#)

pdwReserved: This MUST be set to NULL and MUST be ignored on receipt.

2.2.1.21.2.7.1 COSERVERINFO

The **COSERVERINFO** structure SHOULD NOT be sent and MUST be ignored on receipt.

```
typedef struct _COSERVERINFO {
    DWORD dwReserved1;
    [string] wchar_t* pwszName;
    DWORD* pdwReserved;
    DWORD dwReserved2;
} COSERVERINFO;
```

dwReserved1: This MUST be set to zero and MUST be ignored on receipt.

pwszName: This SHOULD be set to NULL and MUST be ignored by servers. <38>

pdwReserved: This MUST be set to NULL and MUST be ignored on receipt.

dwReserved2: This MUST be set to zero and MUST be ignored on receipt.

2.2.1.21.2.8 ScmReplyInfoData

The **ScmReplyInfoData** structure contains a [customREMOTE_REPLY_SCM_INFO](#) structure.

CLSID_ScmReplyInfo (see section [1.9](#)) is used to identify this property in the CustomHeader.pclsid array.

```
typedef struct tagScmReplyInfoData {
    DWORD* pdwReserved;
    customREMOTE_REPLY_SCM_INFO* remoteReply;
} ScmReplyInfoData;
```

pdwReserved: This MUST be set to NULL and MUST be ignored on receipt.

remoteReply: This MUST specify the **customREMOTE_REPLY_SCM_INFO** for the object exporter of the server object.

2.2.1.21.2.8.1 customREMOTE_REPLY_SCM_INFO

The **customREMOTE_REPLY_SCM_INFO** structure is used to return information about the object exporter, specifically the OXID, RPC bindings, [COMVERSION](#), and IPID of the IRemUnknown interface and the authentication hint of the object exporter.

```
typedef struct _customREMOTE_REPLY_SCM_INFO {
    OXID Oxid;
    DUALSTRINGARRAY* pdsaOxidBindings;
    IPID ipidRemUnknown;
    DWORD authnHint;
    COMVERSION serverVersion;
} customREMOTE_REPLY_SCM_INFO;
```

Oxid: This MUST contain the OXID identifier for the object exporter.

pdsaOxidBindings: This MUST specify the string and security bindings supported by the object exporter and MUST NOT be NULL. The returned string bindings SHOULD contain endpoints.

ipidRemUnknown: This MUST specify the IPID of the object exporter's Remote Unknown object.

authnHint: This SHOULD contain an RPC **authentication level** (see section [2.2.1.26.1](#)) that denotes the minimum authentication level supported by the object exporter. This field MAY be ignored by the client. <39>

serverVersion: This MUST contain the **COMVERSION** of the server.

2.2.1.21.2.9 PropsOutInfo

The **PropsOutInfo** structure represents a collection of interfaces that the object implements and that are returned to the client. If the object does not support a particular interface requested by the client, it also sends an error back using this structure.

CLSID_PropsOutInfo (see section [1.9](#)) is used to identify this property in the CustomHeader.pclsid array.

```
typedef struct tagPropsOutInfo {
    [range (1, MAX_REQUESTED_INTERFACES)]
    DWORD cIfs;
    [size_is(cIfs)] IID* piid;
    [size_is(cIfs)] HRESULT* phresults;
    [size_is(cIfs)] MInterfacePointer** ppIntfData;
} PropsOutInfo;
```

cIfs: This MUST contain the number of interfaces being returned by the server. This value MUST be between 1 and MAX_REQUESTED_INTERFACES (see section [2.2.1.26.3](#)).

piid: This MUST be an array of IIDs identifying the interfaces returned by the server.

phresults: This MUST be an array of status codes indicating the success or failure of each attempt to return an interface requested by the client. For each array location containing a zero value, a non-NULL MInterfacePointer pointer MUST be present in the corresponding location in the ppIntfData array. For each array location containing a negative value, a NULL MUST be present in the corresponding location in the ppIntfData array.

ppIntfData: This MUST be an array of MInterfacePointer pointers containing the OBJREFs for the interfaces returned by the server.

2.2.1.22 REMINTERFACEREF

The **REMINTERFACEREF** structure is passed as a parameter to either [IRemUnknown::RemAddRef \(Opnum 4\)](#) or [IRemUnknown::RemRelease \(Opnum 5\)](#). It specifies the number and type of references that the client requests to be added to (or subtracted from) an interface reference count.

```
typedef struct tagREMINTERFACEREF {
    IPID ipid;
    unsigned long cPublicRefs;
    unsigned long cPrivateRefs;
} REMINTERFACEREF;
```

ipid: This MUST be the IPID of the interface reference count to be modified.

cPublicRefs: This MUST be the number of public references (see section [1.3.6](#)) on the interface identified by IPID being requested by the client.

cPrivateRefs: This MUST be the number of private references (see section [1.3.6](#)) on the interface identified by IPID being requested by the client.

2.2.1.23 REMQIRERESULT

The **REMQIRERESULT** structure is passed as an output parameter from [IRemUnknown::RemQueryInterface](#). It contains the result of the RemQueryInterface and the [STDOBJREF](#) containing the object reference for the queried interface.

```
typedef struct tagREMQIRERESULT {  
    HRESULT hResult;  
    STDOBJREF std;  
} REMQIRERESULT;
```

hResult: This MUST contain zero if the QueryInterface operation was successful. Otherwise this MUST contain a negative value to indicate failure; see section [3.2.1.5.6.1.1](#).

std: If hResult is zero, this MUST contain a STDOBJREF instance that the client may unmarshal and use to make calls on the interface. If hResult contains an error value, this field MUST be ignored.

2.2.1.24 PREMQUIRERESULT

The **PREMQUIRERESULT** type specifies a pointer to a [REMQIRERESULT](#) structure.

This type is declared as follows:

```
typedef [disable_consistency_check] REMQIRERESULT* PREMQUIRERESULT;
```

where **disable_consistency_check** is as specified in [\[MS-RPCE\]](#) section 2.2.4.15

2.2.1.25 REFIPID

The **REFIPID** type specifies a pointer to a const [GUID](#) structure.

This type is declared as follows:

```
typedef const GUID* REFIPID;
```

2.2.1.26 Constant Definitions

2.2.1.26.1 RPC Authentication Level Constants

The following table lists the constants used to specify the protection level for the RPC binding, as specified in [\[MS-RPCE\]](#).

RPC_C_AUTHN_LEVEL_DEFAULT is equivalent to RPC_C_AUTHN_LEVEL_CONNECT.
RPC_C_AUTHN_LEVEL_NONE specifies no security.

Name	Value
RPC_C_AUTHN_LEVEL_DEFAULT	0
RPC_C_AUTHN_LEVEL_NONE	1
RPC_C_AUTHN_LEVEL_CONNECT	2
RPC_C_AUTHN_LEVEL_CALL	3
RPC_C_AUTHN_LEVEL_PKT	4
RPC_C_AUTHN_LEVEL_PKT_INTEGRITY	5
RPC_C_AUTHN_LEVEL_PKT_PRIVACY	6

2.2.1.26.2 RPC Authentication Service Constants

The following table lists the constants used to specify the authentication service for the RPC binding, as specified in [\[MS RPCCE\]](#) section 1.4.

RPC_C_AUTHN_NONE specifies no security. RPC_C_AUTHN_DEFAULT is equivalent to RPC_C_AUTHN_WINNT.

Name	Value
RPC_C_AUTHN_NONE	0
RPC_C_AUTHN_GSS_NEGOTIATE	9
RPC_C_AUTHN_WINNT	10
RPC_C_AUTHN_GSS_KERBEROS	16
RPC_C_AUTHN_DEFAULT	0xFFFFFFFF

2.2.1.26.3 IDL Range Constants

The following table lists the constants used to specify lower and upper limits for various parameters.

Name	Value
MAX_REQUESTED_INTERFACES	0x8000
MAX_REQUESTED_PROTSEQS	0x8000
MIN_ACTPROP_LIMIT	1
MAX_ACTPROP_LIMIT	10

3 Protocol Details

This section specifies the DCOM Remote Protocol using two roles, the client role and the server role. Implementations **MUST** support the client role and **MAY** support the server role. [<40>](#)

In this section, the term object reference **MUST** be regarded as one whose marshaled representation is an [OBJREF](#) that contains a [STDOBJREF](#), unless otherwise specified by an application or a higher-layer protocol.

3.1 Client Details

Clients **SHOULD** support version 5.7 of the DCOM Remote Protocol. [<41>](#)

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The organization is provided to explain how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that specified in this document.

Clients **MUST** maintain the following data elements:

IPID table: A table of entries to interfaces on objects keyed by IPID. Each entry **MUST** contain:

- The IPID of the interface.
- The IID of the interface.
- The OID of the object.
- The OXID of the object exporter.
- The public reference counts of the object reference.
- The private reference counts of the object reference.

OXID table: A table of entries to object exporters known by the client, keyed by OXID. Each entry **MUST** contain:

- The OXID of the object exporter.
- The RPC binding information for the object exporter.
- The IPID of the IRemUnknown interface of the object exporter.
- The Authentication level hint for the object exporter.
- The [COMVERSION](#) of the exporter.

OID table: A table of entries to objects known by the client, keyed by OID. Each entry **MUST** contain:

- The OID of the object.
- A list of IPIDs of the interfaces on the object.
- The OXID of the object exporter.

- An implementation-defined hash of the STRINGBINDING of the saResAddr field contained in the [STDOBJREF](#).
- A Boolean garbage_collection flag that MUST be set to True if the object participates in pinging; see the SORF_NOPING flag in section [2.2.1.17.2](#).

Resolver table: A table of entries to object resolvers known by the client, keyed by a STRINGBINDING hash. Each entry MUST contain:

- A STRINGBINDING hash.
- The DUALSTRINGARRAY of the object resolver.
- The SETID containing the ping set identifier of the object resolver.
- The RPC binding information for the object resolver.

SETID table: A table of entries to **ping sets** referenced by the client, keyed by SETID. Each entry MUST contain:

- The SETID of the ping set.
- A list of OIDs that are in the ping set.
- A sequence number.

3.1.2 Timers

Clients MUST maintain the following timers:

- Global Timer: The client SHOULD maintain a global timer for garbage collection pings. The timer period MUST be at most two minutes and SHOULD be exactly two minutes.

3.1.3 Initialization

There are no protocol initialization steps required on the client beyond the initialization required for all RPC-based protocols, as specified in [\[MS-RPCE\]](#).

3.1.4 Higher-Layer Triggered Events

The operation of DCOM is initiated and subsequently driven by the following higher-layer triggered events:

- A higher-layer protocol or an application creates an object reference.
- The application uses the object reference to make ORPC calls.
- The application acquires and uses object references for additional interfaces on the object.
- The application marshals an object reference.
- The application releases an object reference.

3.1.4.1 Creating Object References

The DCOM client MUST support the following DCOM mechanisms for creating and resolving object references:

- Activation. To use the activation mechanism, the higher-layer application or protocol supplies to the DCOM client a remote server name, the CLSID of the object class, and one or more IIDs of interfaces for which it requires references. The higher-layer application or protocol may also supply non-default values for **security provider**, authentication level, **impersonation level**, SPN, and credentials settings. It is the responsibility of the specification of the higher-layer application or protocol to state such requirements, if any.
- OXID resolution. To use the OXID resolution mechanism, the higher-layer application or protocol supplies to the DCOM client an object reference obtained during unmarshaling parameters during an ORPC call, and the IID of the interface for which it requires a reference.

3.1.4.1.1 Activation

The activation mechanism consists of the following sequence of operations:

1. First, the client MUST use the application-supplied **remote server name** to determine the RPC binding information to be used for the activation request. The client MUST also perform capability negotiation with the server (see section [1.7](#)).
2. Then the client MUST issue the activation request using the application-supplied CLSID and the RPC binding information obtained in the first step.
3. Next, the client MUST update the OXID table.
4. Finally, the client MUST unmarshal the object references returned from the activation request.

3.1.4.1.1.1 Determining RPC Binding Information for Activation

The client MUST call the [IObjectExporter::ServerAlive2](#) method using the first RPC protocol sequence listed in section [3.2.2.3](#), unless the client **CONVERSION** is less than 5.6, in which case the client MUST call the [IObjectExporter::ServerAlive](#) method. The client MUST specify the RPC endpoint information to contain the remote server name on which the application wants to activate the object and the well-known endpoint of the object resolver.

The client MUST NOT specify security on the call.

If the **IObjectExporter::ServerAlive2/IObjectExporter::ServerAlive** call fails with the `RPC_S_UNKNOWN_IF` error (as specified in [\[MS-ERREF\]](#) section 2.2) indicating that the interface is not registered with the endpoint mapper of the server:

- The client MUST perform **dynamic endpoint** resolution as specified in [\[C706\]](#) section [2.2.3](#).
- The client MUST call the **IObjectExporter::ServerAlive2** and **IObjectExporter::ServerAlive** methods again.

If the client calls the **IObjectExporter::ServerAlive2** method, and if the call fails with the `RPC_S_PROCNUM_OUT_OF_RANGE` error (as specified in [\[MS-ERREF\]](#) section 2.2) indicating that the **opnum** is not in range:

- The client MUST pick the RPC binding information used for the current **IObjectExporter::ServerAlive2** call to be used for the activation request.
- The client MUST assume a server **CONVERSION** of 5.1.
- The client SHOULD proceed to issue the activation request, as described in section [3.1.4.1.1.2](#).

Otherwise, if the call fails with any other RPC error:

- The client MUST pick the next RPC protocol sequence listed in section [3.2.2.3](#).
- The client MUST call the **IObjectExporter::ServerAlive2** and **IObjectExporter::ServerAlive** methods again, and MUST treat RPC_S_UNKNOWN_IF and RPC_S_PROCNUM_OUT_OF_RANGE errors as specified above.

If the call succeeds:

- The client MUST pick the RPC binding information used for the current ServerAlive/ServerAlive2 call to be used for the activation request.
- The client SHOULD proceed to issue the activation request as described in section [3.1.4.1.1.2](#).

If the client cannot determine the RPC binding information to be used for the activation request after using all the RPC protocol sequence listed in section [3.2.2.3](#), the client MUST fail the activation request from the DCOM application and SHOULD return RPC_S_SERVER_UNAVAILABLE (as specified in [MS-ERREF] section 2.2).

When the call completes, the client MUST perform capability negotiation as described in section [1.7](#). For the server's COM version, the client MUST use the **COMVERSION** obtained from the **IObjectExporter::ServerAlive2** method. If the **IObjectExporter::ServerAlive2** method failed with the RPC_S_PROCNUM_OUT_OF_RANGE error (as specified in [MS-ERREF] section 2.2), or, if the client used the **IObjectExporter::ServerAlive** method, the client MUST use 5.1 as the server's COM version.

3.1.4.1.1.2 Issuing the Activation Request

The client MUST proceed to issue an activation request by:

- Selecting the security parameters.
- Selecting the activation interface.
- Selecting the activation request parameters.

The client SHOULD specify security on the activation call. The client SHOULD use the [SECURITYBINDING](#) contained in the [DUALSTRINGARRAY](#) returned from the **IObjectExporter::ServerAlive2** method to pick a security provider that is common to both the client and the object resolver, and that meets the client's security requirements.

The client MUST specify the default values for the following security settings:

The client MUST specify the credentials requested by the application, if supplied; otherwise, it MUST specify the credentials of the security principal of the application that is issuing the activation request as the default credentials for the activation call.

The client MUST specify the authentication level requested by the application, if one was supplied; otherwise, it MUST specify a default authentication level that is obtained in an implementation-specific manner. [<42>](#)

The client MUST specify the impersonation level requested by the application, if one was supplied; otherwise, it MUST specify a default impersonation level of at least RPC_C_IMPL_LEVEL_IMPERSONATE (see [\[MS-RPCE\]](#) section 2.2.1.1.9). [<43>](#)

When using Kerberos and SPNEGO security providers, the client MUST specify the SPN requested by the application, if one was supplied; otherwise, it MUST specify an SPN of "RPCSS/<remote server

name>", where <remote server name> is replaced by the remote server name passed by the higher-layer application or protocol to the activation request.

The client MUST select the interface used to make the activation request as follows:

- If the server's COM version, as determined in section [3.1.4.1.1.1](#), is less than 5.6, the client MUST select the IActivation interface.
- Otherwise, the client MUST select the IRemoteSCMAActivator interface if the client requires the additional activation capabilities supported by the IRemoteSCMAActivator interface over the IActivation interface.
- Otherwise, the client MUST select the IActivation interface.

If the client selects the IActivation interface, it MUST specify the Mode parameter of the [RemoteActivation](#) method as follows:

- If the DCOM application requires an interface on the class factory object, the client MUST set the Mode parameter to -1.
- Otherwise, the client MUST set the Mode parameter to 0.

If the client selects the IRemoteSCMAActivator interface, it MUST pick an Activation method as follows:

- If the DCOM application requires an interface on the class factory object, the client MUST call the [RemoteGetClassObject](#) method.
- Otherwise, the client MUST call the [RemoteCreateInstance](#) method.

The client MUST specify the remaining parameters common to both interfaces as follows:

- The client MUST set the **CONVERSION** in the [ORPCTHIS](#) parameter to the value negotiated while determining the RPC binding information (see section [3.1.4.1.1.1](#)).
- The client MUST specify the CLSID supplied by the DCOM application.
- The client MUST supply an array of RPC Protocol Sequence Identifiers that the client's object resolver listens on, as specified in section [3.2.2.3](#). The array SHOULD contain all such identifiers (as opposed to only a subset of them).
- The client MUST specify an array of IIDs supplied by the DCOM application.

When the activation call returns successfully, the following data MUST be returned to the client:

- An [HRESULT](#) indicating the overall result of the activation. If this is not success, the client MUST return the HRESULT to the DCOM application.
- The OXID and the RPC bindings of the object exporter.
- The IPID of the [IRemUnknown](#) interface of the object exporter.
- The **CONVERSION** of the server.
- An authentication hint for use in subsequent ORPCs to the object.
- An array of HRESULTS, one for each interface, that the client requested. If an element of this array is not S_OK, the client MUST consider that the object reference for the corresponding interface is not obtained.

- An array of object references to the interfaces on the object.

3.1.4.1.1.3 Updating the Client OXID Table after Activation

The client MUST proceed to update its object exporter identifier (OXID) table as follows:

- The client MUST look up the OXID entry in the OXID table using the OXID value returned from the activation request. If the OXID entry is not found, the client MUST add an entry to the OXID table as follows:
- The client MUST set the OXID, RPC bindings, interface pointer identifier (IPID), authentication level hint, and the [CONVERSION](#) of the OXID entry to the corresponding values returned from the activation request.

For each object reference returned from the activation request for which the corresponding status code indicates success, the client MUST unmarshal the object reference, as specified in the next section.

3.1.4.1.2 Unmarshaling an Object Reference

The client MUST use NDR to unmarshal an OBJREF from a [MInterfacePointer](#) in the RPC PDU body. The Unmarshaling mechanism consists of the following sequence of operations:

- The client MUST extract the IID and the [OBJREF_STANDARD](#) from the [OBJREF](#) if the flags field specifies OBJREF_STANDARD. Otherwise, the client MUST supply the OBJREF to the application and/or higher-layer protocols to unmarshal [OBJREF_HANDLER](#), [OBJREF_EXTENDED](#), and [OBJREF_CUSTOM](#) OBJREF types.
- Next, the client MUST extract the [STDOBJREF](#) and the [DUALSTRINGARRAY](#) from the OBJREF_STANDARD.
- Next, the client MUST look up the OXID entry in the OXID table using the OXID from the **STDOBJREF**. If the table entry is not found:
 - The client MUST determine the RPC binding information to be used for the OXID resolution request, as specified in section [3.1.4.1.2.1](#).
 - Next, the client MUST issue the OXID resolution request to obtain the RPC binding information of the object exporter, as specified in section [3.1.4.1.2.2](#).
- Next, the client MUST update its tables, as specified in section [3.1.4.1.2.3](#).
- Finally, the client MUST compare the IID in the OBJREF with the IID specified by the application when unmarshaling the object reference (see section [4.5](#)). If the IIDs are not equal:
 - First, the client SHOULD acquire an object reference of the IID specified by the application; see section [3.1.4.4.3](#). The client MAY instead report an error to the application. [<44>](#)
 - Next, the client MUST release the object reference unmarshaled from the OBJREF (see section [3.1.4.4.2](#)).

3.1.4.1.2.1 Determining RPC Binding Information for OXID Resolution

The client MUST call the [IObjectExporter::ServerAlive2](#) method as follows, unless the client CONVERSION is less than 5.6, in which case the client MUST call the [IObjectExporter::ServerAlive](#) method:

- The client MUST specify the RPC protocol sequence corresponding to the wTowerId in the first STRINGBINDING of the saResAddr contained in the STDOBJREF of the object reference.
- The client MUST specify the RPC endpoint information to contain the aNetworkAddr in the first STRINGBINDING of the saResAddr contained in the STDOBJREF of the object reference and the well-known endpoint of the object resolver.
- The client MUST NOT specify security on the call.

If the **IObjectExporter::ServerAlive2/IObjectExporter::ServerAlive** call fails with the RPC_S_UNKNOWN_IF error (as specified in [MS-ERREF] section 2.2) indicating that the interface is not registered with the endpoint mapper:

- The client MUST perform dynamic endpoint resolution as specified in [C706] section 2.2.3.
- The client MUST call the **IObjectExporter::ServerAlive2** and **IObjectExporter::ServerAlive** methods again.

If the client calls the **IObjectExporter::ServerAlive2** method, and if the call fails with the RPC_S_PROCNUM_OUT_OF_RANGE error (as specified in [MS-ERREF] section 2.2) indicating that the opnum is not in range:

- The client MUST pick the RPC binding information used for the current **IObjectExporter::ServerAlive2** call to be used for the OXID resolution request.
- The client SHOULD proceed to issue the OXID resolution as specified in section 3.1.4.1.2.2.

Otherwise, if the call fails with any other RPC error:

- The client MUST specify the RPC protocol sequence corresponding to the wTowerId in the next STRINGBINDING of the saResAddr contained in the STDOBJREF of the object reference.
- The client MUST specify the RPC endpoint information to contain the aNetworkAddr in the next STRINGBINDING of the saResAddr contained in the STDOBJREF of the object reference and the well-known endpoint of the object resolver.
- The client MUST make the **IObjectExporter::ServerAlive2/IObjectExporter::ServerAlive** call again, and MUST treat RPC_S_UNKNOWN_IF and RPC_S_PROCNUM_OUT_OF_RANGE errors as specified above.

Otherwise, if the call succeeds:

- The client MUST pick the RPC binding information used for the current **IObjectExporter::ServerAlive2** call to be used for the OXID resolution request.
- The client SHOULD proceed to issue the OXID resolution, as specified in section 3.1.4.1.2.2.

If the client cannot determine the RPC binding information to be used for OXID resolution after using all the RPC protocol sequences available in the [STRINGBINDING](#) structure contained in the saResAddr field, the client MUST fail the unmarshaling of the object reference and SHOULD return OR_INVALID_OXID (as specified in [MS-ERREF] section 2.2) to the DCOM application.

3.1.4.1.2.2 Issuing the OXID Resolution Request

The client MUST proceed to issue an OXID resolution request by:

- Selecting the security parameters.

- Selecting the OXID resolution request parameters.

The client SHOULD specify security on the OXID resolution request. The client SHOULD use the [SECURITYBINDING](#) structure contained in the **saResAddr** field of the OBJREF to pick a security provider that is common to both the client and the object resolver, and that meets the client's security requirements.

The client MUST specify the values for the following security settings:

The client MUST specify the credentials of the security principal of the application or higher-level protocol that is issuing the OXID resolution request as the credentials for the call.

The client MUST specify an authentication level of at least `RPC_C_AUTHN_LEVEL_CONNECT` (see [\[MS-RPCE\]](#) section 2.2.1.1.8).[<45>](#)

The client MUST specify an impersonation level of at least `RPC_C_IMPL_LEVEL_IDENTIFY` (see [\[MS-RPCE\]](#) section 2.2.1.1.9).[<46>](#)

When using Kerberos and SPNEGO security providers, the client MUST specify an SPN of "RPCSS/<remote server name>", where <remote server name> is the network address used to create the RPC binding information, as specified in section [3.1.4.1.2.1](#).

The client MUST call the [ResolveOxid2](#) method of the [IObjectExporter](#) interface to make the OXID resolution request.

If the **ResolveOxid2** method fails with the `RPC_S_PROCNUM_OUT_OF_RANGE` error (as specified in [\[MS-ERREF\]](#) section 2.2), the client MUST retry by calling the [ResolveOxid](#) method. In this case, it MUST assume that the [CONVERSION](#) structure of the server specifies version 5.1.

The client MUST specify parameters to the OXID resolution request as follows:

- The client MUST specify the OXID of the object exporter contained in the object reference supplied by the DCOM application.
- The client MUST supply an array of RPC protocol sequence identifiers that the client's object resolver listens on, as specified in section [3.2.2.3](#). The array SHOULD contain all such identifiers (as opposed to only a subset of them).

When OXID resolution completes successfully, the following data MUST be returned to the client.

- The OXID bindings of the object exporter.
- IPID of the [IRemUnknown](#) interface of the object exporter.
- The **CONVERSION** of the server in the case of **ResolveOxid2**, or 5.1 in the case of **ResolveOxid**.
- An authentication hint for use in subsequent ORPCs to the server.

3.1.4.1.2.3 Updating Client Tables after Unmarshaling

3.1.4.1.2.3.1 Updating the OXID Table after Unmarshaling

The client MUST proceed to update its tables in the following sequence:

- The client MUST look up the OXID entry in the OXID table using the OXID in the [STDOBJREF](#) of the object reference. If the OXID entry is not found, the client MUST add an entry to the OXID table as follows:
 - The client MUST set the OXID to the OXID in the **STDOBJREF** of the object reference.
 - The client MUST set the RPC bindings, IPID, Authentication-level hint, and the [CONVERSION](#) of the OXID entry to the corresponding values returned from the activation request.

3.1.4.1.2.3.2 Updating the OID/IPID/Resolver Tables after Unmarshaling

- The client MUST look up the interface pointer identifier (IPID) entry in the IPID table using the IPID contained in the [STDOBJREF](#) of the object reference.
 - If the entry is found, the client MUST add the public reference count from the **STDOBJREF** to the public reference count in the IPID entry.
 - If the **STDOBJREF** contains a public reference count of 0, the client MUST obtain additional references on the interface using the steps specified in section [3.1.4.4.1](#).
 - If the entry is not found, the client MUST add an entry to the IPID table as follows:
 - The client MUST set the IPID, OXID, and OID in the IPID entry to the corresponding values contained in the **STDOBJREF**.
 - The client MUST set the interface identifier (IID) to the IID in the OBJREF.
 - The client MUST set the public reference count to that contained in the **STDOBJREF**.
 - If the **STDOBJREF** contains a public reference count of zero, the client MUST obtain additional references on the interface using the steps specified in section [3.1.4.4.1](#).
 - The client MUST set the private reference counts to zero.
- The client MUST look up the OID entry in the OID table using the OID contained in the **STDOBJREF** of the object reference.
 - If the OID entry is found, and if the IPID is not present in the IPID list of the OID entry, the client MUST add the IPID to the IPID list of the OID entry.
 - If the OID entry is not found, the client MUST add an entry to the OID table as follows:
 - The client MUST set the OID to the value of the OID contained in the **STDOBJREF** of the object reference.
 - The client MUST add the IPID contained in the **STDOBJREF** of the object reference to the IPID list.
 - The client MUST set the garbage_collection flag to True if **STDOBJREF** does not contain the SORF_NOPING flag.
 - The client MUST store the [STRINGBINDING](#) hash of the **saResAddr** field in the **STDOBJREF** to the OID entry.
 - The client MUST look up the STRINGBINDING hash in the Resolver table. If the table entry is not found, the client MUST add an entry to the Resolver table, including the hash key and the RPC binding information used for the activation request or the OXID resolution request. The client MUST set the SETID to 0 in the Resolver table entry.

3.1.4.2 ORPC Invocations

To make an ORPC call, a DCOM application supplies to the DCOM client an IPID to reference a specific interface on an object, a method number (opnum), and a list of arguments to the method. The DCOM application may also supply nondefault values for security provider, authentication level, impersonation level, SPN, and credentials settings. It is the responsibility of the specification of the higher-layer protocol to state such requirements, if any.

When an ORPC call is made, the DCOM client MUST perform the following sequence:

1. It MUST look up the object exporter information in the client tables.
2. It MUST perform capability negotiation.
3. It MUST specify security settings for the ORPC.
4. It MUST make the ORPC request.

The client MUST use the IPID specified by the client application to look up the IPID entry in the IPID table. The client MUST then look up the OXID entry to obtain the [DUALSTRINGARRAY](#) that contains the RPC binding information, the [COMVERSION](#), and the authentication-level hint of the object exporter.

The client MUST perform capability negotiation using the **COMVERSION** of the server, as specified in section [1.7](#).

The client MUST use the [SECURITYBINDING](#) contained in the DUALSTRINGARRAY for the object exporter bindings to pick a security provider that is common to both the client and the object exporter, and that meets the client's security requirements. If the SECURITYBINDING structure is empty, the client MUST NOT specify any security on the call.

If the client specifies security on the call, it MUST specify the default values for the following security settings:

- The client MUST specify the credentials requested by the application, if supplied; otherwise, it MUST specify the credentials of the security principal of the application that is issuing the ORPC request as the default credentials for the call.
- The client MUST specify the authentication level requested by the application, if one was supplied; otherwise, it MUST specify a value that is the higher value of the client's authentication level value, obtained in an implementation-specific manner, and the authentication-level hint of the object exporter. [<47>](#)
- The client MUST specify the impersonation level requested by the application, if one was supplied; otherwise, it MUST specify a default impersonation level that is obtained in an implementation-specific manner. [<48>](#)
- The client MUST specify the SPN requested by the application, if one was supplied; otherwise, it MUST specify the **aPrincName** field in the SECURITYBINDING packet contained in the DUALSTRINGARRAY of the object exporter bindings, if the **aPrincName** field is non-empty; otherwise, if the **aPrincName** field is empty, the client MUST NOT specify an SPN.

The client MUST initiate the ORPC as follows:

- The client MUST specify the IID from the IPID entry in the RPC interface UUID field.
- The client MUST specify the RPC interface version as 0.0.

- The client MUST specify the application-supplied RPC opnum of the method on the interface.
- The client MUST specify the application-supplied IPID in the object UUID field.
- The client MUST specify the [ORPCTHIS](#) as the first implicit parameter in the ORPC request. In particular:
 - The client MUST set the **COMVERSION** to the negotiated version from activation or OXID resolution.
 - The client MUST set the **cid** to the CID of the current ORPC. If the client is currently executing an incoming ORPC, the client MUST set the **cid** of the outgoing ORPC to be the same as the **cid** in the **ORPCTHIS** of the incoming ORPC. If the client is not executing an incoming ORPC, the client MUST specify a new CID. For details, see section [1.3.5](#).
 - The client MAY specify the **extensions** field if it needs to send out-of-band data to the object. [<49>](#)
- The client MUST marshal ORPC parameters of object reference types; see section [3.1.4.3](#).

[<50>](#)

The client MUST process the ORPC response as follows:

- The [ORPCTHAT](#) structure will be returned as the first implicit parameter of the ORPC response. In particular:
 - The client MAY process the **extensions** field if it needs to receive out-of-band data from the object. [<51>](#)
- If an object reference is returned as a parameter from the ORPC, the client MUST unmarshal it; see section [3.1.4.1.2](#).

3.1.4.3 Marshaling an Object Reference

When a DCOM application sends an object reference in an ORPC invocation, it MUST supply the interface pointer identifier (IPID). To marshal the object reference, the client MUST do the following:

- It MUST look up the IPID entry using the IPID.
- It MUST look up the OID entry using the OID from the IPID entry.
- It MUST look up the Resolver table entry using the [STRINGBINDING](#) hash from the OID entry.
- It MUST create an [STDOBJREF](#) instance as follows:
 - It MUST set the flags field to SORF_NOPING if the garbage_collection flag in the OID entry is True. Otherwise, it MUST set the flags field to 0.
 - If the IPID entry contains more than one public reference count, it SHOULD decrement the public reference count of the IPID entry and set cPublicRefs to 1.
 - Otherwise, it MUST set cPublicRefs to 0.
 - It MUST set the oxid field in the **STDOBJREF** instance to the value of the OXID from the IPID entry.

- It MUST set the oid field in the **STDOBJREF** instance to the value of the OID from the IPID entry.
- It MUST set the ipid field in the **STDOBJREF** instance to the value of the IPID specified by the DCOM application.
- It MUST create an OBJREF_STANDARD instance as follows:
 - It MUST set the std field to the **STDOBJREF** instance created above.
 - It MUST set the saResAddr field to the [DUALSTRINGARRAY](#) from the Resolver table entry.
- It MUST create an OBJREF instance as follows:
 - It MUST set the signature field to 0x574f454d.
 - It MUST set the flags field to OBJREF_STANDARD.
 - It MUST set the iid field to the IID from the IPID entry.
 - It MUST set the u_objref to the OBJREF_STANDARD created above.
- It MUST use NDR to marshal the OBJREF into a [MInterfacePointer](#) that is placed in the RPC PDU body.

3.1.4.4 Managing Object Lifetime

If the client unmarshals an object reference with an [STDOBJREF](#) instance that has a public reference count of 0, the client MUST request additional reference counts for the object reference to keep the object alive. For a description of how an **STDOBJREF** instance may be created with 0 public reference counts, see section [3.1.4.3](#).

3.1.4.4.1 Requesting Reference Counts on an Interface

To request additional reference counts, the client MUST perform the following sequence of operations:

- It MUST look up the OXID entry for the OXID specified in the [STDOBJREF](#) instance.
- It MUST use the IPID of the [IRemUnknown](#) interface of the object exporter contained in the OXID entry to issue an ORPC call (see section [3.1.4.2](#)) on the [RemAddRef](#) method of the **IRemUnknown** interface.

The client MUST specify the following when making the call:

- An [REMINTERFACEREF](#) structure containing:
 - The IPID contained in the **STDOBJREF**.
 - The public reference count that the client requires.
 - The client MAY specify a private reference count if the application requires exclusive references. [<52>](#)

When the call returns successfully, the client MUST do the following:

- It MUST increment the public reference count in the IPID entry for the IPID contained in the **STDOBJREF**.

- It MUST increment the private reference count in the IPID entry for the IPID contained in the **STDOBJREF**, if the client requested private reference counts.

3.1.4.4.2 Releasing Reference Counts on an Interface

When the DCOM application is finished using the object reference, the client MUST perform the following sequence of operations:

- It MUST look up the IPID entry for the IPID specified by the application.
- Using the OXID contained in the IPID entry, it MUST look up the OXID entry.
- Using the OID contained in the IPID entry, it MUST look up the OID entry.
- It MUST use the IPID of the [IRemUnknown](#) interface of the object exporter contained in the OXID entry to issue an ORPC call, see section [3.1.4.2](#), on the [RemRelease](#) method of the **IRemUnknown** interface.

The client MUST specify the following when making the call:

- A [REMINTERFACEREF](#) structure containing:
 - The IPID specified by the application.
 - The number of public reference counts contained in the IPID entry.
 - The number of private reference counts contained in the IPID entry. [<53>](#)

When the call returns successfully, the client MUST perform the following:

- It MUST remove the IPID entry from the IPID table.
- It MUST remove the IPID from the IPID list in the OID entry.
- If the IPID list of the OID entry is empty, it MUST remove the OID entry from the OID table.

3.1.4.4.3 Acquiring Additional Interfaces on the Object

When the DCOM application requests references to additional interfaces on the object, the client MUST perform the following sequence of operations:

- It MUST look up the IPID entry for the IPID specified by the application.
- It MUST look up the OXID entry for the OXID specified in the [STDOBJREF](#) instance.
- It MUST use the IPID of the [IRemUnknown](#) interface of the object exporter contained in the OXID entry to issue an ORPC call (see section [3.1.4.2](#)) on the [RemQueryInterface](#) method of the **IRemUnknown** interface.

The client MUST specify the following information when making the call:

- The IPID specified by the application.
- An array of IIDs supplied by the client application.
- The number of public reference counts requested on the new interfaces.

When the call returns successfully, the client MUST perform the following for each entry in the [REMOIRESULT](#) array:

- If the status code contains zero, the client MUST unmarshal the corresponding **STDOBJREF** and return the object reference to the DCOM application (see section [3.1.4.1.2](#)).

3.1.5 Message Processing Events and Sequencing Rules

Not applicable for clients.

3.1.6 Timer Events

When the global timer expires, the client MUST make ping calls.

3.1.6.1 Pinging

For each entry in the Resolver table:

- If the SETID is 0, the client MUST add a SETID entry as follows:
 - It MUST search the OID table and form a list of OIDs with the same [STRINGBINDING](#) hash as the Resolver table entry, and add the list to the SETID entry. If the `garbage_collection` flag (see the `SORF_NOPING` flag in section [2.2.1.17.2](#)) is set to FALSE in the OID entry, the OID MUST not be added to the list.
 - It MUST set the sequence number to 0.
 - It MUST call the [ComplexPing](#) method of the [IObjectExporter](#) using the RPC binding information in the Resolver table entry. The client MUST specify:
 - The SETID as 0.
 - The sequence number as 0.
 - The OID list as the *AddToSet* parameter.
 - The *DelFromSet* parameter as NULL.
 - When the call completes, the client MUST do the following:
 - It MUST set the SETID returned from the call in the SETID entry.
 - It MUST increment the sequence number in the SETID entry.
 - It MUST set the SETID in the Resolver table entry.
- If the SETID is not NULL, the client MUST do the following:
 - It MUST look up the SETID entry using the SETID from the Resolver table entry. If the entry is found:
 - It MUST search the OID table and form a list of OIDs with the same `STRINGBINDING` hash as the Resolver table entry and compare the list to the OID list in the SETID entry.
 - If the sets are the same, the client SHOULD call the [SimplePing](#) method of the **IObjectExporter** interface using the RPC binding information from the Resolver table entry. The client MUST specify the SETID in the call.

- If the sets are NOT the same, the client MUST do the following:
 - It MUST increment the sequence number.
 - It MUST call the **ComplexPing** method of the **IObjectExporter** interface using the RPC binding information from the Resolver table entry. The client MUST specify:
 - The SETID from the SETID entry.
 - The OIDs added since the last ping period in the *AddToSet* parameter.
 - The OIDs removed since the last ping period in the *DelFromSet* parameter.
 - The sequence number from the SETID entry.
 - When the call completes, the client MUST store the new OID set in the SETID entry.
- If the new OID set is empty, the client MUST do the following:
 - It MUST remove the SETID entry from the SETID table.
 - It MUST remove Resolver table entry.

For both the **SimplePing** and the **ComplexPing** calls, the client MUST specify the values of the following security settings:

- The client MUST specify the authentication service associated with the RPC Binding stored in the Resolver table.
- The client MUST specify the credentials of the security principal of the application or higher-level protocol that is issuing the ping request as the credentials for the call.
- The client MUST specify an authentication level of at least `RPC_C_AUTHN_LEVEL_CONNECT` (see [\[MS-RPCE\]](#) section 2.2.1.1.8).<54>
- The client MUST specify an impersonation level of at least `RPC_C_IMPL_LEVEL_IDENTIFY` (see [\[MS-RPCE\]](#) section 2.2.1.1.9).<55>

When using Kerberos and SPNEGO security providers, the client MUST specify an SPN of "RPCSS/<remote server name>", where <remote server name> is the network address used to create the RPC binding information stored in the Resolver table (see section [3.1.4.1.2.3.2](#)).

3.1.7 Other Local Events

Not applicable for clients.

3.2 Object Server Details

An object server MUST contain exactly one object resolver and zero or more object exporters. The object resolver and the object exporters communicate by way of an implementation-specific mechanism.

3.2.1 Object Exporter Details

3.2.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The organization is provided to explain how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that specified in this document.

Object exporters MUST maintain the following data elements:

Authentication Level: The authentication level of the object exporter.

Permissions: An implementation-specific set of permissions of who can access the object exporter.

IPID table: A table of entries to interfaces on objects, keyed by IPID or IID. Each entry MUST contain:

- The IPID of the interface.
- The IID of the interface.
- The OID of the object.
- The OXID of the object exporter.
- The public reference counts of the object reference.
- A list of private reference counts, one per client identity.
- A pointer to an application defined state for the object's implementation of the interface.

OID table: A table information about objects referenced by the client, keyed by OID or object pointer. Each entry MUST contain:

- The OID of the object.
- The OXID of the object exporter.
- A list of IPIDs of the interfaces on the object.
- The time of the last ORPC invocation on the OID.
- An object pointer to an implementation-specific application state that represents the object.
- An implementation-defined hash of the [STRINGBINDING](#) of the **saResAddr** field contained in the [STDOBJREF](#).
- A Boolean garbage_collection flag that MUST be set to True if the object participates in pinging; see the SORF_NOPING flag in section [2.2.1.17.2](#).

Resolver table: See section [3.1.1](#).

SETID table: See section [3.1.1](#).

OXID table: See section [3.1.1](#).

3.2.1.2 Timers

Each object exporter instance MUST maintain a global timer for pinging, similar to those maintained by clients; see section [3.1.2](#).

3.2.1.3 Initialization

Object exporters MUST listen by way of RPC protocols described in section [3.2.2.3](#) and as specified in [\[C706\]](#) section [3.1.20](#) (`rpc_server_use_protseq`). Because listening is potentially expensive, such initialization SHOULD be delayed until an object reference is marshaled from the object exporter.

The object exporter MUST listen on an RPC protocol before returning the object exporter bindings that specify that RPC protocol sequence identifier in the response to the activation or OXID resolution requests.

The object exporter MUST obtain its OXID from the resolver.

The object exporter MUST obtain the OID of the Remote Unknown object from the resolver.

The object exporter MUST obtain the ping period from the resolver; for details, see section [3.2.2.2](#).

Each object exporter instance MUST create an IPID entry for its [IRemUnknown](#) interface. If an object exporter instance is at [CONVERSION](#) 5.6 or above, it MUST also create an IPID entry for the [IRemUnknown2](#) interface. An object exporter instance MUST create its IPID entry as follows:

- It MUST allocate an IPID and set it in the IPID entry.
- It MUST set the IID in the IPID entry to the IID for the **IRemUnknown** interface or the **IRemUnknown2** interface.
- It MUST instruct RPC to listen on the **IRemUnknown** interface or the **IRemUnknown2** interface, as specified in [\[C706\]](#) section [3.1.20](#) (`rpc_server_register_if`).
- It MUST set the object pointer in the entry to the object pointer of the object exporter's implementation of the **IRemUnknown** interface or the **IRemUnknown2** interface.
- It MUST set the OID and the OXID in the IPID entry to the corresponding values obtained from the resolver.
- It MUST add the IPID entry to the IPID table.

3.2.1.4 Higher-Layer Triggered Events

When a higher-layer protocol or an application specifies the values of the authentication level and the permissions needed to access the object exporter, the object server MUST store them in the corresponding fields specified in section [3.2.1.1](#). It is the responsibility of the specification of the application or the higher-layer protocol to state such requirements, if any.

3.2.1.5 Message Processing Events and Sequencing Rules

3.2.1.5.1 Marshaling an Object

The object exporter MUST perform the following operations to marshal an object into an OBJREF of OBJREF_STANDARD type when returning from an ORPC. In this case, the application or higher-layer protocol MUST supply an object pointer and an IID; see section [4.5](#).

- The object exporter MUST look up the OID entry in the OID table using the object pointer.

- If the OID entry is not found, the object exporter MUST create an entry as follows:
 - The object exporter MUST contact the object resolver to allocate an OID and set it in the OID entry.
 - The object exporter MUST initialize the IPID list in the OID entry to NULL.
 - The object exporter MUST set the object pointer in the OID entry to the object pointer supplied by the application.
- The object exporter MUST look up the IPID entry in the IPID list of the OID entry using the IID specified by the application.
 - If the IPID entry is not found, the object exporter MUST create an IPID entry as follows:
 - The object exporter MUST allocate an IPID and set it in the IPID entry.
 - The object exporter MUST set the OID in the IPID entry to the OID obtained from the object resolver.
 - The object exporter MUST set the OXID in the IPID entry to the OXID of the object exporter.
 - The object exporter MUST set the IID in the IPID entry to the IID specified by the application; see section [4.5](#).
 - The object exporter MUST set the public reference count in the IPID entry to an initial value. The initial value MUST be 0 or more, and SHOULD be 5.
 - The object exporter MUST set the private reference count to 0.
 - The object exporter MUST instruct RPC to listen on the interface designated by the IID, as specified in [\[C706\]](#) section [3.1.20](#) (rpc_server_register_if).
 - Set the object pointer in the IPID entry to the object pointer of the application's state that implements the interface specified by the IID. The application state is implementation-specific.
 - The object exporter MUST add the IPID to the IPID list of the OID entry.
 - Otherwise, the object exporter MUST increment the public reference count in the IPID entry by the initial value.
- The object exporter MUST set the last ORPC invocation time in the OID entry to the current time.
- The object exporter MUST construct a [STDOBJREF](#) instance for the object as follows:
 - The object exporter SHOULD set the flags field to 0. [<56>](#)
 - The object exporter MUST set cPublicRefs to the initial value.
 - The object exporter MUST set the IPID and the OID from the respective IPID and OID entries.
 - The object exporter MUST set the OXID to the OXID of the object exporter.
- The object exporter MUST create an OBJREF_STANDARD instance as follows:
 - The object exporter MUST set the **std** field to the constructed **STDOBJREF** instance.

- The object exporter MUST set the **saResAddr** field to the DUALSTRING array of the object resolver.
- The object exporter MUST create an OBJREF instance as follows:
 - The object exporter MUST set the **signature** field to 0x574f454d.
 - The object exporter MUST set the **flags** field to OBJREF_STANDARD.
 - The object exporter MUST set the **iid** field to the IID supplied by the application.
 - The object exporter MUST set the **u_objref** field to the OBJREF_STANDARD instance.
- The object exporter MUST use NDR to marshal the OBJREF into an [MInterfacePointer](#) that is placed in the RPC PDU body.

Application and higher-layer protocols MAY also instruct the object exporter to marshal OBJREF instances of type [OBJREF_HANDLER](#), [OBJREF_EXTENDED](#), and [OBJREF_CUSTOM](#). In this case, the object exporter MUST create an OBJREF, as specified above, with the exception that it MUST set the flags field to the corresponding flag type of the OBJREF, and the **u_objref** field to the corresponding OBJREF instance.

3.2.1.5.2 Marshaling an Object Reference

The object exporter MUST perform the following operations to marshal an object reference when returning from an ORPC invocation. In this case, the application or higher-layer protocol MUST supply the IPID of the object reference.

- The object exporter MUST proceed as a client to marshal the object reference. Specifically, it MUST follow the steps specified in section [3.1.4.3](#) to marshal the object reference.

3.2.1.5.3 Unmarshaling an Object Reference

The object exporter MUST perform the following operations to unmarshal an object reference when receiving an ORPC invocation. The object exporter MUST use NDR to unmarshal an OBJREF from a [MInterfacePointer](#) in the RPC PDU body. The unmarshaling mechanism consists of the following sequence of operations:

- If the flags field specifies OBJREF_STANDARD, the object exporter MUST extract the IID and the OBJREF_STANDARD instance from the OBJREF instance. Otherwise, the object exporter MUST supply the OBJREF instance to the application and/or higher-layer protocols to unmarshal [OBJREF_HANDLER](#), [OBJREF_EXTENDED](#), and [OBJREF_CUSTOM](#) OBJREF types.
- Next, the object exporter MUST extract the [STDOBJREF](#) instance and the [DUALSTRINGARRAY](#) instance from the OBJREF_STANDARD instance.
- The object exporter MUST compare the OXID in the **STDOBJREF** with the OXID of the object exporter. If they are the same, it MUST do the following:
 - The object exporter MUST look up the IPID entry in the IPID table.
 - If the IPID entry is not found, the object exporter MUST return E_NOINTERFACE (as specified in [\[MS-ERREF\]](#) section 2.1).
 - If the IPID entry is found, the object exporter MUST decrement the public reference count in the IPID entry by the cPublicRefs in the **STDOBJREF**. If the public reference count in the IPID entry becomes zero, the object exporter MUST remove the IPID entry from the IPID table.

- The object exporter MUST look up the OID entry using the OID in the **STDOBJREF**.
- The object exporter MUST return the object pointer in the OID entry to the application.
- Otherwise, the object exporter MUST proceed as a client. To unmarshal the object reference, see section [3.1.4.1.2](#).

3.2.1.5.4 ORPC Invocations

Object exporters MUST process ORPC invocations. An ORPC invocation is the receipt of an RPC request with an object UUID field set to a non-GUID_NULL value, which is interpreted as an IPID. The ORPC invocation specifies a method number (opnum), an **ORPCTHIS**, and a list of arguments to the method.

Upon receipt of an ORPC request, the object exporter MUST perform the following:

- It MUST perform capability negotiation as discussed in section [1.7](#); specifically, it MUST return RPC_E_VERSION_MISMATCH (as specified in [\[MS-ERREF\]](#) section 2.1) if the client has a higher COM minor version or a different COM major version.
- If the object exporter requires security on the incoming ORPC, the object exporter MUST check the security settings on the incoming ORPC as follows:
 - If the authentication level of the incoming ORPC is lower than the authentication level of the object exporter, the object exporter MUST return E_ACCESSDENIED (as specified in [\[MS-ERREF\]](#) section 2.1).
 - The authentication level of the object exporter MUST be the value specified by the application or the higher level protocol (see section [3.2.1.4](#)). If the value is not specified by the application or higher-level protocol, it MUST be the default value obtained in an implementation-specific manner. [<57>](#)

The object exporter MUST check if the client has the permissions to make the ORPC request. If the permissions are not granted, the object exporter MUST return E_ACCESSDENIED (as specified in [\[MS-ERREF\]](#) section 2.1).

The permissions of the object exporter MUST be the value specified by the application or the higher-level protocol (see section [3.2.1.4](#)). If the value is not specified by the application or the higher-level protocol, it MUST be the default value obtained in an implementation-specific manner. [<58>](#)

- It MUST use the **cid** field of the **ORPCTHIS** parameter to determine if the incoming ORPC belongs to the same causality chain as that of a currently executing outgoing ORPC (see section [1.3.5](#)).
 - If yes, the object exporter MUST process the incoming ORPC.
 - Otherwise, the object exporter MAY delay the processing of the incoming ORPC until the currently executing outgoing ORPC completes. It may do this to prevent re-entrancy in the application. [<59>](#)
- It MAY process the extensions fields in the **ORPCTHIS** implicit input parameter. [<60>](#)
- It MUST look up the IPID entry identified by the IPID in the RPC UUID field. If the IPID entry is not found, the object exporter MUST return RPC_E_DISCONNECTED (as specified in [\[MS-ERREF\]](#) section 2.1).

- It MUST unmarshal input parameters that are object reference types, as specified in section [3.2.1.5.3](#).
- It MUST invoke the application-specified state in the IPID entry using the opnum and *input* parameters specified in the RPC request.
- It MUST return the output parameters from the application in the RPC response.
- It MUST look up the OID entry and set the last ORPC invocation time to the current time.
- It MAY return the **extensions** field in the [ORPCTHAT](#) implicit output parameter. [<61>](#)
- If an object is returned as a parameter from the ORPC, the object exporter MUST marshal the object (see section [3.2.1.5.1](#)).
- If an object reference is returned as a parameter from the ORPC, the object exporter MUST marshal the object reference (see section [3.2.1.5.2](#)).

[<62>](#)

3.2.1.5.5 Lazy Protocol Registration

If the object exporter delays RPC protocol initialization, the object exporter MUST respond to requests from the object resolver to listen on RPC protocols, as specified in [\[C706\]](#) section [3.1.20](#) (*rpc_server_use_protseq*), as part of activation and OXID resolution requests.

3.2.1.5.6 IRemUnknown Interface

Object exporters MUST implement the **IRemUnknown** interface. The **IRemUnknown** interface is used by clients to manage object lifetime and to acquire additional references on the object.

Methods in RPC Opnum Order

Method	Description
RemQueryInterface	Acquires standard object references to additional interfaces on the object. Opnum: 3
RemAddRef	Requests that a specified number of reference counts be incremented on a specified number of interfaces on the object. Opnum: 4
RemRelease	Requests that a specified number of reference counts be decremented on a specified number of interfaces on an object. Opnum: 5

The methods MUST NOT throw exceptions.

3.2.1.5.6.1 IRemUnknown Methods

All object exporters MUST implement a Remote Unknown object that supports the [IRemUnknown](#) interface. Clients MUST check for the *RPC_S_UNKNOWN_IF* error (as specified in [\[MS-ERREF\]](#) section 2.2) to determine if the object exporter supports the **IRemUnknown** interface.

This is an **ORPC** interface, not a native **RPC** interface.

3.2.1.5.6.1.1 IRemUnknown::RemQueryInterface (Opnum 3)

This **RemQueryInterface (Opnum 3)** method acquires standard object references (see section [2.2.1.17.1](#)) to additional interfaces on the object.

```
HRESULT RemQueryInterface(  
    [in] REFIID riid,  
    [in] unsigned long cRefs,  
    [in] unsigned short cIids,  
    [in, size_is(cIids)] IID* iids,  
    [out, size_is(cIids)] REMQIRERESULT* ppQIResults  
);
```

riid: This MUST specify an IPID that identifies the interface on the object to be queried for more interfaces.

cRefs: This MUST specify the number of public reference counts requested on the new interfaces.

cIids: This MUST specify the number of IIDs supplied in the *iids* parameter and returned in the *ppQIResults* parameter.

iids: This MUST specify an array of IIDs for which the client requests object references.

ppQIResults: This MUST contain an array of [REMQIRERESULT](#) structures containing the results of the QueryInterface on the identified object.

When processing this ORPC call, the object exporter MUST do the following:

- It MUST look up the IPID entry of the interface specified by the *riid* parameter. If the IPID entry is not found, it MUST return `RPC_E_INVALID_OBJECT` as specified in [\[MS-ERREF\]](#) section 2.1.
- It MUST look up the OID entry for the IPID.
- It MUST set the last ORPC invocation time in the OID entry to the current time.
- For each IID requested by the client in the *iids* array:
 - It MUST find the IPID of the IID in the IPID list in the OID entry.
 - If the entry is found, it MUST increment the public reference counts by *cRefs*, the number of references requested by the client.
 - If the entry is not found, it MUST use the application-defined state to determine, in an implementation-specific manner, if the object supports the requested IID. If it does not, it MUST set `E_NOINTERFACE` (as specified in [\[MS-ERREF\]](#) section 2.1) in the **hResult** field of the corresponding **REMQIRERESULT** array.

If the object implements the requested IID, the object exporter MUST create an IPID entry as follows:

- It MUST allocate an IPID and set it in the IPID entry.
- It MUST set the IID in the entry to the IID specified by the client.
- It MUST set the OID in the IPID entry to the OID in the OID entry.

- It MUST set the public reference count to cRefs, the number of references requested by the client.
- It MUST set the private reference count to 0.
- It MUST instruct RPC to listen on the interface designated by the IID, as specified in [\[C706\]](#) section [3.1.20](#) (rpc_server_register_if).
- It MUST set the object pointer in the entry to the object pointer of the application's state that implements the interface specified by the IID.
- It MUST add the IPID entry to the IPID list in the OID.
- It MUST add the IPID entry to the IPID table.
- The object exporter MUST return the STDOBJREF representing the object reference in the corresponding **REMQUIRERESULT** array element as follows:
 - It MUST set the **flags** field to 0.
 - It MUST set the **cPublicRefs** field to cRefs.
 - It MUST set the IPID and the OID from the IPID and OID entries.
 - It MUST set the OXID to the OXID of the object exporter.
- The object exporter MUST set the corresponding HRESULT field to a success code of zero.

3.2.1.5.6.1.2 IRemUnknown::RemAddRef (Opnum 4)

The **RemAddRef (Opnum 4)** method requests that a specified number of reference counts be incremented on a specified number of interfaces on the object.

```
HRESULT RemAddRef (
    [in] unsigned short cInterfaceRefs,
    [in, size_is(cInterfaceRefs)] REMINTERFACEREF InterfaceRefs[],
    [out, size_is(cInterfaceRefs)]
        HRESULT* pResults
);
```

cInterfaceRefs: This MUST specify the number of elements sent in the *InterfaceRefs* parameter and returned in the *pResults* parameter.

InterfaceRefs: This MUST specify an array of [REMINTERFACEREF](#) structures, each of which specifies the number of public and private references to be added to the interface identified by the IPID.

pResults: This MUST contain an array of HRESULTs specifying the respective success or failure of the RemAddRef operation for each **REMINTERFACEREF** element.

When processing this ORPC call, the object exporter MUST do the following:

For each **REMINTERFACEREF** element in the InterfaceRefs array argument:

- It MUST look up **REMINTERFACEREF.ipid** in the IPID table to find the IPID entry. If the entry is not found, it MUST set the return code in the corresponding pResults array to CO_E_OBJNOTREG (as specified in [\[MS-ERREF\]](#) section 2.1).

- If the entry is found, the object exporter MUST do the following:
 - It MUST increment the public reference count in the IPID entry by **REMINTERFACEREF.cPublicRefs**.
 - It MUST increment the private reference count in the IPID entry by **REMINTERFACEREF.cPrivateRefs**.
 - It SHOULD associate the private reference counts exclusively for use by the client.
 - It MUST set the return code in the corresponding pResults array to a success code of zero. [<63>](#)

3.2.1.5.6.1.3 IRemUnknown::RemRelease (Opnum 5)

The **RemRelease (Opnum 5)** method requests that a specified number of reference counts be decremented on a specified number of interfaces on an object.

```
HRESULT RemRelease(
    [in] unsigned short cInterfaceRefs,
    [in, size_is(cInterfaceRefs)] REMINTERFACEREF InterfaceRefs[]
);
```

cInterfaceRefs: This MUST specify the number of elements in the *InterfaceRefs* parameter.

InterfaceRefs: This MUST specify an array of [REMINTERFACEREF](#) structures, each of which specifies the number of public and private references to be released on the interface identified by the IPID.

When processing this ORPC call, the object exporter MUST do the following:

For each **REMINTERFACEREF** element in the *InterfaceRefs* array argument:

- The object exporter MUST look up **REMINTERFACEREF.ipid** in the IPID table to find the IPID entry. If the entry is found, the object exporter MUST do the following:
 - It MUST decrement the public reference count in the IPID entry by **REMINTERFACEREF.cPublicRefs**. If **REMINTERFACEREF.cPublicRefs** is greater than the public reference count in the IPID entry, the object exporter MUST set the public reference count in the IPID entry to 0.
 - It MUST decrement the private reference count associated with the client in the IPID entry by **REMINTERFACEREF.cPrivateRefs**. If **REMINTERFACEREF.cPrivateRefs** is greater than the private reference count in the IPID entry, the object exporter MUST set the private reference count in the IPID entry to 0. [<64>](#)
- If the public and private reference counts are zero, the object exporter MUST do the following:
 - It MUST remove the IPID entry from the IPID table.
 - It MUST remove the IPID from the list of IPIDs in the OID table.
 - It MUST remove the application-specific state of the object that implements the interface from the IPID entry.

- It MUST instruct RPC to stop listening on the interface, as specified in [\[C706\]](#) section [3.1.20](#) (`rpc_server_unregister_if`).
- If the list of IPIDs in the OID entry is empty, it MUST remove the OID entry from the OID table.
- It MUST remove the application-specific state of the object from the OID entry.
- It MUST contact the object resolver to release the OID.

3.2.1.5.7 IRemUnknown2 Interface

If the server reports a COM version of 5.6 or greater, the object exporter Remote Unknown object MUST support the **IRemUnknown2** interface. Clients MUST check for the `RPC_S_UNKNOWN_IF` error (as specified in [\[MS-ERREF\]](#) section 2.2) to determine if the object exporter supports the **IRemUnknown2** interface.

This is an **ORPC** interface, not a native **RPC** interface.

Methods in RPC Opnum Order

Method	Description
RemQueryInterface2	Acquires standard object references (see section 2.2.1.17.1) to additional interfaces on the object, marshaled as MInterfacePointer . Opnum: 6

The method listed above MUST NOT throw an exception.

3.2.1.5.7.1 IRemUnknown2 Methods

3.2.1.5.7.1.1 IRemUnknown2::RemQueryInterface2 (Opnum 6)

The **RemQueryInterface2 (Opnum 6)** method acquires standard object references (see section [2.2.1.17.1](#)) to additional interfaces on the object, marshaled as an [MInterfacePointer](#) structure.

```
HRESULT RemQueryInterface2(
    [in] REFIPID ripid,
    [in] unsigned short cIids,
    [in, size_is(cIids)] IID* iids,
    [out, size_is(cIids)] HRESULT* phr,
    [out, size_is(cIids)] PMInterfacePointerInternal* ppMIF
);
```

ripid: This MUST specify an IPID that identifies the interface on the object in the object exporter that is queried for more interfaces.

cIids: This MUST specify the number of elements in the *iids*, *phr*, and *ppMIF* parameters.

iids: This MUST specify an array of IIDs for which the client requests object references.

phr: This MUST contain an array of HRESULTs specifying the respective success or failure of each query operation.

ppMIF: This MUST contain an array of **MInterfacePointer** structures containing the results of each query operation.

When processing this ORPC call, the object exporter MUST perform the same operations as it does for [IRemUnknown::RemQueryInterface](#), except that it MUST return an OBJREF for each object reference in the **MInterfacePointer** array, instead of an **STDOBJREF**.

3.2.1.5.8 IUnknown Interface

The **IUnknown** interface enables clients to retrieve pointers to other interfaces on a given object through the **QueryInterface** method, and to manage the existence of the object through the **IUnknown::AddRef** and **IUnknown::Release** methods. All other COM interfaces are inherited, directly or indirectly, from **IUnknown**. Therefore, the three methods in **IUnknown** reserve opnums 0 through 2, inclusive, for every interface.

IUnknown is a local-only interface.

The Universally Unique Identifier (UUID) for this interface is {00000000-0000-0000-C000-000000000046}.

Methods in RPC Opnum Order

Method	Description
Opnum0NotUsedOnWire	QueryInterface Reserved for local use. Opnum: 0
Opnum1NotUsedOnWire	AddRef Reserved for local use. Opnum: 1
Opnum2NotUsedOnWire	Release Reserved for local use. Opnum: 2

In the table above, the term "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined [<65>](#) since it does not affect interoperability.

All methods MUST NOT throw exceptions.

3.2.1.6 Timer Events

3.2.1.6.1 Pinging

When the global timer expires, the object exporter MUST make ping calls, as specified in section [3.1.6.1](#).

3.2.1.6.2 Object Reclamation

The object exporter MUST respond to object reclamation requests from the object resolver when the object resolver determines, by the pinging mechanism, that the client of the object is dead or unreachable; see section [3.2.2.6](#). When processing this request, the object exporter MUST do the following:

- It MUST look up the OID entry for the object. If found:
 - It MUST check the last ORPC invocation time to see if an ORPC request has arrived in the last ping period.
 - If yes, the object exporter SHOULD perform no further action.
 - If not, it MUST do the following:
 - For each IPID in the IPID list in the OID entry:
 - It MUST remove the IPID entry from the IPID table.
 - It MUST remove the IPID from the list of IPIDs in the OID table.
 - It MUST remove the application-specific state of the object that implements the interface from the IPID entry.
 - It MUST instruct RPC to stop listening on the interface, as specified in [\[C706\]](#) section [3.1.20](#) (rpc_server_unregister_if).
 - It MUST remove the application-specific state of the object in the OID entry.
 - It MUST remove the OID from the OID entry.
 - It MUST contact the object resolver to release the OID.

3.2.1.7 Other Local Events

Not applicable to object exporters.

3.2.2 Object Resolver Details

3.2.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The organization is provided to explain how the protocol behaves.

This document does not mandate that an implementation adhere to this model, as long as its external behavior is consistent with that specified in this document.

Object resolvers MUST maintain the following data elements:

CLSID table: A table of entries to object exporters managed by the resolver, keyed by CLSID. Each entry MUST contain:

- The CLSID of the object class.
- A pointer to an application-defined state for implementation of a class factory interface.
- The OXID entry of the object exporter of the object class.

OXID table: A table of entries to object exporters managed by the resolver, keyed by OXID. Each entry MUST contain:

- The OXID of the object exporter.

- The RPC binding information for the object exporter.
- The IPID of the [IRemUnknown](#) interface of the object exporter.
- An authentication-level hint for the object exporter.
- The [CONVERSION](#) of the object exporter.

SETID table: A table of ping set entries allocated by the resolver, keyed by SETID. Each entry MUST contain:

- The SETID of the ping set.
- A list of OIDs that are in the ping set.
- A sequence number.
- A ping set timer.

OID table: A table of OID entries allocated by the resolver, keyed by OID. Each entry MUST contain:

- An OID.
- A count of the number of SETID entries that reference the OID entry.
- The OXID of the object exporter of the OID.

3.2.2.2 Timers

Object resolvers MUST maintain the following timers:

- A ping set. The resolver MUST maintain a timer for each ping set whose SETID it returns to clients. The ping set timer period MUST be at least 3 times the ping period and SHOULD be exactly 3 times the ping period. The ping period MUST be at most 2 minutes and SHOULD be exactly 2 minutes.

3.2.2.3 Initialization

The DCOM Remote Protocol assumes that the object resolver is running and reachable (see section [1.9](#)) before the protocol is used. The object resolver MAY listen on a variety of RPC protocols. [<66>](#)

3.2.2.4 Higher-Layer Triggered Events

Not applicable to object resolvers.

3.2.2.5 Message Processing Events and Sequencing Rules

3.2.2.5.1 IObjectExporter Methods

IObjectExporter is the interface used for OXID resolution, pinging, and server aliveness tests. All object resolvers MUST support the **IObjectExporter** interface. Clients MUST check for the RPC_S_UNKNOWN_IF error (as specified in [\[MS-ERREF\]](#) section 2.2) to determine if the object resolver supports the **IObjectExporter** interface.

This is a native RPC interface, not an **ORPC** interface.

Methods in RPC Opnum Order

Method	Description
ResolveOxid	Returns the bindings and Remote Unknown IPID for an object exporter. Opnum: 0
SimplePing	Performs a ping of a previously allocated ping set to maintain the reference counts on the objects referred to by the set. Opnum: 1
ComplexPing	Invoked to create or modify a ping set, to ping a ping set, or to perform a combination of these operations in one invocation. Opnum: 2
ServerAlive	Invoked by clients to test the aliveness of a server using a given RPC protocol. Opnum: 3
ResolveOxid2	Returns the bindings and Remote Unknown IPID for an object exporter, and the CONVERSION of the object server. Opnum: 4
ServerAlive2	Introduced with version 5.6 of the DCOM Remote Protocol. Extends the ServerAlive method and returns string and security bindings for the object resolver. Opnum: 5

The methods MUST NOT throw exceptions.

3.2.2.5.1.1 IObjectExporter::ResolveOxid (Opnum 0)

The **ResolveOxid** method returns the bindings and Remote Unknown IPID for an object exporter.

```
[idempotent] error_status_t ResolveOxid(
    [in] handle_t hRpc,
    [in] OXID* pOxid,
    [in] unsigned short cRequestedProtseqs,
    [in, ref, size_is(cRequestedProtseqs)]
        unsigned short arRequestedProtseqs[],
    [out, ref] DUALSTRINGARRAY** ppdsaOxidBindings,
    [out, ref] IPID* pipidRemUnknown,
    [out, ref] DWORD* pAuthnHint
);
```

hRpc: This MUST specify an RPC binding handle, as specified in [\[MS-RPCE\]](#) section 3.2.2.3.1.

pOxid: This MUST specify an OXID identifying an object exporter.

cRequestedProtseqs: This MUST contain the number of elements in the arRequestedProtseqs array.

arRequestedProtseqs: This MUST contain an array of RPC protocol sequence identifiers supported by the client.

ppdsaOxidBindings: This MUST contain the string and security bindings supported by the object exporter and MUST NOT be NULL. The returned string bindings SHOULD contain endpoints.

pipidRemUnknown: This MUST contain the IPID of the object exporter Remote Unknown object.

pAuthnHint: This SHOULD contain an RPC authentication level (see section [2.2.1.26.1](#)) that indicates the minimum authentication level supported by the object exporter, which MAY be ignored by the client. [<67>](#)

When processing this call:

- The object resolver MAY compare the client's credentials against a set of implementation-specific permissions. [<68>](#) If the permissions do not allow access to the client credentials, the object resolver MUST return ERROR_ACCESS_DENIED (as specified in [MS-ERREF] section [2.2](#)).
- The object resolver MUST look up the OXID entry using the OXID specified by the client. If the entry is not found, it MUST return OR_INVALID_OXID (as specified in [\[MS-ERREF\]](#) section 2.2).
- If the entry is found and the RPC binding information in the entry is not present or does not contain the RPC protocol sequence that the client specified, the object resolver MUST instruct the object exporter to listen on the RPC protocol, as specified in [\[C706\]](#) section [3.1.20](#) (rpc_server_use_protseq), and set the RPC binding string of the object exporter in the OXID entry.
- The object resolver MUST return the following information to the client:
 - The RPC binding strings of the object exporter.
 - The authentication hint of the object exporter.
 - The IPID of the [IRemUnknown](#) interface of the object exporter.

3.2.2.5.1.2 IObjectExporter::SimplePing (Opnum 1)

The **SimplePing** method performs a ping of a previously allocated ping set to maintain the reference counts on the objects referred to by the set.

```
[idempotent] error_status_t SimplePing(  
    [in] handle_t hRpc,  
    [in] SETID* pSetId  
);
```

hRpc: This MUST specify an RPC binding handle as specified in [\[MS-RPCE\]](#) section 3.2.2.3.1 .

pSetId: MUST specify a SETID of the ping set to ping. This parameter MUST specify a SETID previously returned from a call to [IObjectExporter::ComplexPing](#).

When processing this call:

- The object resolver MAY compare the client's credentials against a set of implementation-specific permissions. [<69>](#) If the permissions do not allow access to the client credentials, the object resolver MUST return ERROR_ACCESS_DENIED (as specified in [MS-ERREF] section [2.2](#)).
- The object resolver MUST look up the SETID entry of the SETID designated by the client. If the entry is not found, return OR_INVALID_SET (as specified in [\[MS-ERREF\]](#) section 2.2).
- If the entry is found, the object resolver MUST reset the ping set timer for the SETID entry.

3.2.2.5.1.3 IObjectExporter::ComplexPing (Opnum 2)

The **ComplexPing (Opnum 2)** method is invoked to create or modify a ping set, to ping a ping set, or to perform a combination of these operations in one invocation.

```
[idempotent] error_status_t ComplexPing(  
    [in] handle_t hRpc,  
    [in, out] SETID* pSetId,  
    [in] unsigned short SequenceNum,  
    [in] unsigned short cAddToSet,  
    [in] unsigned short cDelFromSet,  
    [in, unique, size_is(cAddToSet)]  
        OID AddToSet[],  
    [in, unique, size_is(cDelFromSet)]  
        OID DelFromSet[],  
    [out] unsigned short* pPingBackoffFactor  
);
```

hRpc: This MUST specify an RPC binding handle as specified in [\[MS-RPCE\]](#) section 3.2.2.3.1 .

pSetId: This MUST specify the SETID of the ping set to ping. If the SETID specified by the client is zero, the object resolver MUST return the SETID of a new ping set containing the OIDs specified in AddToSet. If the SETID specified by the client is not zero, the object exporter MUST not change the SETID.

SequenceNum: This MUST specify a sequence number shared between the client and the object resolver.

cAddToSet: This MUST specify the number of OIDs in the AddToSet array.

cDelFromSet: This MUST specify the number of OIDs in the DelFromSet array.

AddToSet: This MUST specify an array of OIDs to add to the set.

DelFromSet: This MUST specify an array of OIDs to remove from the set.

pPingBackoffFactor: This MUST contain a hint to reduce ping load on the server. Servers MAY set the hint to an implementation-specific value. Clients MAY choose to treat this as zero always. [<70>](#)

When processing this call:

- The object resolver MAY compare the client's credentials against a set of implementation-specific permissions. [<71>](#) If the permissions do not allow access to the client credentials, the object resolver MUST return ERROR_ACCESS_DENIED (as specified in [\[MS-ERREF\]](#) section 2.2).
- If the client specifies a nonzero SETID, the object resolver MUST look up the SETID entry for the SETID. If the entry is not found, the object resolver MUST return OR_INVALID_SET (as specified in [\[MS-ERREF\]](#) section 2.2).
- If the entry is found, and if it has a sequence number greater than that specified by the client, the object resolver MUST take no further action and return success.
- Otherwise, the object resolver MUST do the following:
 - For each OID in the AddToSet list:

- It MUST look up the OID in the OID list in the SETID entry.
- If not found:
 - It MUST look up the OID entry in the OID table. If found:
 - It MUST increment the reference count in the OID entry.
 - It MUST add the OID to the OID list in the SETID entry.
 - Otherwise, it MUST return OR_INVALID_OID ([MS-ERREF] section [2.2](#)).
- For each OID in the DelFromSet list:
 - It MUST look up the OID in the OID list in the SETID entry.
 - If found:
 - It MUST look up the OID entry in the OID table. If found:
 - It MUST decrement the reference count in the OID entry.
 - It MUST remove the OID from the OID list in the SETID entry.
- It MUST reset the ping timer in the SETID entry.
- It MUST set the sequence number to the value specified by the client.
- If the client specifies a NULL SETID, it MUST add a new SETID entry as follows:
 - It MUST allocate a SETID.
 - It MUST set the sequence number to the value specified by the client.
 - For each OID in the AddToSet list, it MUST do the following:
 - It MUST look up the OID entry. If the entry is found:
 - It MUST increment the reference count in the OID entry.
 - It MUST add the OID to the OID list in the SETID entry.
 - It MUST start the ping set timer. It MUST initialize the timer, as specified in section [3.2.2.2](#).
 - It MUST return the SETID to the client.

3.2.2.5.1.4 IObjectExporter::ServerAlive (Opnum 3)

The **ServerAlive (Opnum 3)** method is used by clients to test the aliveness of the server using a given RPC protocol. If it returns without an error, the server is assumed to be reachable.

```
[idempotent] error_status_t ServerAlive(
    [in] handle_t hRpc
);
```

hRpc: This MUST specify an RPC binding handle as specified in [\[MS-RPCE\]](#) section 3.2.2.3.1 .

When processing this call:

- The object resolver MUST NOT check if the client has the permissions to make this call.
- The object resolver MUST return an RPC success code of 0.

3.2.2.5.1.5 IObjectExporter::ResolveOxid2 (Opnum 4)

The **ResolveOxid2** method returns the bindings and Remote Unknown IPID for an object exporter, and the [CONVERSION](#) of the object server. This method was introduced with version 5.2 of the DCOM Remote Protocol.

```
[idempotent] error_status_t ResolveOxid2(
    [in] handle_t hRpc,
    [in] OXID* pOxid,
    [in] unsigned short cRequestedProtseqs,
    [in, ref, size_is(cRequestedProtseqs)]
        unsigned short arRequestedProtseqs[],
    [out, ref] DUALSTRINGARRAY** ppdsaOxidBindings,
    [out, ref] IPID* pipidRemUnknown,
    [out, ref] DWORD* pAuthnHint,
    [out, ref] CONVERSION* pComVersion
);
```

hRpc: This MUST specify an RPC binding handle as specified in [\[MS-RPCE\]](#) section 3.2.2.3.1 .

pOxid: This MUST specify an OXID identifying an object exporter.

cRequestedProtseqs: This MUST contain the number of elements in the arRequestedProtseqs array.

arRequestedProtseqs: This MUST contain an array of RPC protocol sequence identifiers supported by the client.

ppdsaOxidBindings: This MUST contain the string and security bindings supported by the object exporter and MUST NOT be NULL. The returned string bindings SHOULD contain endpoints.

pipidRemUnknown: This MUST contain the IPID of the object exporter Remote Unknown object.

pAuthnHint: This SHOULD contain an RPC authentication level (see section [2.2.1.26.1](#)) that denotes the minimum authentication level supported by the object exporter. [<72>](#)

pComVersion: This MUST contain the CONVERSION of the object exporter. For details, see section [2.2.1.5](#).

When processing this call, the object resolver MUST perform the same operations that it does for the [IObjectExporter::ResolveOxid](#) method. It MUST also return the object exporter's **CONVERSION** from the OXID entry of the object exporter.

3.2.2.5.1.6 IObjectExporter::ServerAlive2 (Opnum 5)

The **ServerAlive2 (Opnum 5)** method was introduced with version 5.6 of the DCOM Remote Protocol. This method extends the **ServerAlive** method. It returns string and security bindings for the object resolver, which allows the client to choose the most appropriate, mutually compatible settings.


```
[idempotent] error_status_t ServerAlive2(
    [in] handle_t hRpc,
    [out, ref] COMVERSION* pComVersion,
    [out, ref] DUALSTRINGARRAY** ppdsaOrBindings,
    [out, ref] DWORD* pReserved
);
```

hRpc: This MUST specify an RPC binding handle as specified in [\[MS-RPCE\]](#) section 3.2.2.3.1 .

pComVersion: This MUST contain the [COMVERSION](#) of the object resolver; see section [2.2.1.5](#).

ppdsaOrBindings: MUST contain the string and security bindings of the object resolver. The returned string bindings MUST NOT contain endpoints.

pReserved: MUST contain zero and MUST be ignored by the recipient.

When processing this call:

- The object resolver MUST NOT check if the client has the permissions to make this call.
- The object resolver MUST return the object resolver **COMVERSION** and the [DUALSTRINGARRAY](#) containing its RPC bindings to the client.

3.2.2.5.1.7 Allocating and Deleting OID Entries

When the object exporter contacts the object resolver to allocate an OID, see section [3.2.1.5.1](#), the object resolver MUST allocate an OID entry as follows:

- It MUST allocate an OID and set it in the OID entry.
- It MUST set the OXID of the object exporter in the OID entry.
- It MUST set the SETID reference count to 0.

The object resolver MUST remove the OID entry from the OID table when contacted by the object exporter; see [IRemUnknown::RemRelease \(Opnum 5\) \(section 3.2.1.5.6.1.3\)](#).

3.2.2.5.1.8 Allocating OXID Entries

When the object exporter contacts the object resolver to allocate an OXID (see section [3.2.1.3](#)), the object resolver MUST allocate an OXID entry as follows:

- It MUST allocate an OXID and set in the entry.
- It MUST set the RPC binding of the object exporter to NULL.
- It MUST set the **COMVERSION** to that of the object exporter.
- It MUST set the authentication level hint to that of the object exporter.

3.2.2.5.2 IActivation and IRemoteSCMAActivator Methods

3.2.2.5.2.1 IActivation Methods

IActivation is the DCOM Remote Protocol remote activation interface supported on all versions of the DCOM Remote Protocol. All object resolvers MUST support the **IActivation** interface. Clients MUST check for the RPC_S_UNKNOWN_IF error (as specified in [\[MS-ERREF\]](#) section 2.2) to determine if the object resolver supports the **IActivation** interface.

This is a native **RPC** interface, not an **ORPC** interface.

Methods in RPC Opnum Order

Method	Description
RemoteActivation	Used by clients to request the activation of an object. Opnum: 0

The method above MUST NOT throw an exception.

3.2.2.5.2.2 IRemoteSCMAActivator Methods

IRemoteSCMAActivator is another remote activation interface of the DCOM Remote Protocol. All object exporters that report a [COMVERSION](#) of 5.6 or greater MUST support this interface. Clients MUST check for the RPC_S_UNKNOWN_IF error (as specified in [\[MS-ERREF\]](#) section 2.2) to determine if the object resolver supports the **IRemoteSCMAActivator** interface.

This is a native **RPC** interface, not an **ORPC** interface.

Methods in RPC Opnum Order

Method	Description
DummyQueryInterfaceIRemoteSCMAActivator	Deprecated. Opnum: 0
DummyAddRefIRemoteSCMAActivator	Deprecated. Opnum: 1
DummyReleaseIRemoteSCMAActivator	Deprecated. Opnum: 2
RemoteGetClassObject	Used by clients to create an object reference for the class factory object. Opnum: 3
RemoteCreateInstance	Used by clients to create an object reference for the actual object. Opnum: 4

The methods MUST NOT throw exceptions.

3.2.2.5.2.2.1 IRemoteSCMAActivator:: DummyQueryInterfaceIRemoteSCMAActivator (Opnum 0)

The **DummyQueryInterfaceIRemoteSCMAActivator (Opnum 0)** method is deprecated and SHOULD NOT be invoked. [<73>](#)

3.2.2.5.2.2.2 IRemoteSCMAActivator:: DummyAddRefIRemoteSCMAActivator (Opnum 1)

The **DummyAddRefIRemoteSCMAActivator (Opnum 1)** method is deprecated and SHOULD NOT be invoked. [<74>](#)

3.2.2.5.2.2.3 IRemoteSCMAActivator:: DummyReleaseIRemoteSCMAActivator (Opnum 2)

The **DummyReleaseIRemoteSCMAActivator (Opnum 2)** method is deprecated and SHOULD NOT be invoked. [<75>](#)

3.2.2.5.2.3 IActivation::RemoteActivation, IRemoteSCMAActivator::RemoteGetClassObject, IRemoteSCMAActivator::RemoteCreateInstance

These three methods all perform the same conceptual operations and so are specified here together.

When processing this call:

- The object resolver MAY compare the client's credentials against a set of implementation-specific permissions. [<76>](#) If the permissions do not allow access to the client credentials, the object resolver MUST return E_ACCESSDENIED (as specified in [\[MS-ERREF\]](#) section 2.1).
- The object resolver MUST use the [CONVERSION](#) of the client specified in the [ORPCTHIS](#) parameter to perform capability negotiation as described in section [1.7](#). In particular, the object resolver MUST return RPC_E_VERSION_MISMATCH (as specified in [\[MS-ERREF\]](#) section 2.1) if the client has a higher minor COM version or a different COM major version.
- The object resolver MUST look up the CLSID entry for the CLSID specified by the client. If the entry is not found:
 - The object resolver MUST start an object exporter for the CLSID using an implementation-specific mechanism.
 - The object resolver MUST create the CLSID entry specifying the CLSID, the OXID entry of the exporter and the application-specific state of the class factory interface for the CLSID.
- The object resolver MUST invoke the application-specific state of the class factory interface, in an implementation-specific manner, to create the object in its object exporter and marshal an object reference (see section [3.1.4.3](#)) for each IID specified by the client.
- If the RPC binding information in the OXID entry of the object exporter of the object is not present or does not contain the RPC protocol sequence that the client specified, The object resolver MUST instruct the object exporter to listen on the RPC protocol, as specified in [\[C706\]](#) section [3.1.20](#) (rpc_server_use_protseq).
- The object resolver MUST return the following:
 - An array of object references, one for each IID.

- The IPID of the [IRemUnknown](#) interface of the object exporter.
- The **COMVERSION** of the object exporter.
- The RPC bindings of the object exporter.

3.2.2.5.2.3.1 IActivation:: RemoteActivation (Opnum 0)

The **RemoteActivation (Opnum 0)** method is used by clients to request the activation of an object. It returns the bindings, the IPID for the Remote Unknown, and the [COMVERSION](#) of the object exporter that hosts the object.

```
error_status_t RemoteActivation(
    [in] handle_t hRpc,
    [in] ORPCTHIS* ORPCthis,
    [out] ORPCTHAT* ORPCthat,
    [in] GUID* Clsid,
    [in, string, unique] wchar_t* pwszObjectName,
    [in, unique] MInterfacePointer* pObjectStorage,
    [in] DWORD ClientImpLevel,
    [in] DWORD Mode,
    [in, range(1, MAX_REQUESTED_INTERFACES)]
        DWORD Interfaces,
    [in, unique, size_is(Interfaces)]
        IID* pIIDs,
    [in, range(0, MAX_REQUESTED_PROTSEQS)]
        unsigned short cRequestedProtseqs,
    [in, size_is(cRequestedProtseqs)]
        unsigned short aRequestedProtseqs[],
    [out] OXID* pOxid,
    [out] DUALSTRINGARRAY** ppdsaOxidBindings,
    [out] IPID* pipidRemUnknown,
    [out] DWORD* pAuthnHint,
    [out] COMVERSION* pServerVersion,
    [out] HRESULT* phr,
    [out, size_is(Interfaces), disable_consistency_check]
        MInterfacePointer** ppInterfaceData,
    [out, size_is(Interfaces), disable_consistency_check]
        HRESULT* pResults
);
```

hRpc: This MUST specify an RPC binding handle as specified in [\[MS-RPCE\]](#) section 3.2.2.3.1 .

ORPCthis: This MUST specify an ORPCTHIS. The **COMVERSION** field SHOULD contain the negotiated version, as described in section [1.7](#). The extensions field MUST be set to NULL.

ORPCthat: This MUST contain an [ORPCTHAT](#). The extensions field MUST be set to NULL.

Clsid: This MUST specify the CLSID of the object to be created.

pwszObjectName: This MAY contain a string to be used to initialize the object. [<77>](#)

pObjectStorage: This MAY contain a marshaled OBJREF to be used to initialize the object. [<78>](#)

ClientImpLevel: This MUST contain an implementation-specific value that MUST be ignored on receipt. [<79>](#)

Mode: If the activation is for a class factory reference, this parameter MUST be 0xFFFFFFFF. Otherwise, it MUST be 0, except when the client specifies an initialization string in `pwszObjectName`. If it does, this field MAY contain an implementation-specific value. [<80>](#)

Interfaces: This MUST contain the number of elements in `pIIDs`. This value MUST be between 1 and `MAX_REQUESTED_INTERFACES`; see section [2.2.1.26.3](#).

pIIDs: This MUST be an array of requested IIDs on the object to be created.

cRequestedProtseqs: This MUST contain the number of elements in `aRequestedProtseqs`. This value MUST be between 1 and `MAX_REQUESTED_PROTSEQS` (see section [2.2.1.26.3](#)).

aRequestedProtseqs: This MUST specify an array of RPC protocol sequence identifiers that the client supports.

pOxid: This MUST contain an OXID value identifying the object exporter containing this object.

ppdsaOxidBindings: This MUST contain the string and security bindings supported by the object exporter and MUST NOT be NULL. The returned string bindings SHOULD contain endpoints.

pipidRemUnknown: This MUST contain the IPID of the object exporter Remote Unknown object.

pAuthnHint: This SHOULD contain an RPC authentication level, see section [2.2.1.26.1](#), that denotes the minimum authentication level supported by the server. This MAY be ignored by the client. [<81>](#)

pServerVersion: This MUST contain the **CONVERSION** of the object exporter. For details, see section [2.2.1.5](#).

phr: This MUST contain the result of the activation. A value of 0 indicates success, a negative value indicates failure.

ppInterfaceData: This MUST contain an array of [MInterfacePointer](#) structures containing the results for each requested interface.

pResults: If the `phr` parameter contains 0, this MUST contain an array of HRESULTs containing the results of querying the object for each interface in `pIIDs`. A value of 0 indicates success, a negative value indicates failure. If the `phr` parameter contains a negative value, this MUST contain an array of zeros, one for each interface in `pIIDs`.

When processing this call, the object resolver MUST perform the operations specified in section [3.2.2.5.2.3](#).

3.2.2.5.2.3.2 IRemoteSCMAActivator:: RemoteGetClassObject (Opnum 3)

The **RemoteGetClassObject (Opnum 3)** method is used by clients to create an object reference for the class factory object.

```
HRESULT RemoteGetClassObject(  
    [in] handle_t hRpc,  
    [in] ORPCTHIS* orpcthis,  
    [out] ORPCTHAT* orpcthat,  
    [in,unique] MInterfacePointer* pActProperties,  
    [out] MInterfacePointer** ppActProperties
```

);

hRpc: This MUST specify an RPC binding handle as specified in [\[MS-RPCE\]](#) section 3.2.2.3.1 .

orpcthis: This MUST specify an ORPCTHIS. The [CONVERSION](#) field SHOULD contain the negotiated version as specified in section [2.2.1.5](#). The extensions field MUST be set to NULL.

orpcthat: This MUST contain an ORPCTHAT. The extensions field MUST be set to NULL.

pActProperties: This MUST specify an [MInterfacePointer](#) that MUST contain an OBJREF_CUSTOM with a CLSID field set to CLSID_ActivationPropertiesIn (section [1.9](#)) and a **pObjectData** field that MUST contain an activation properties binary large object (BLOB) (section [2.2.1.21](#)). The **iid** field of the [OBJREF](#) portion of the structure MUST be set to a value that an implementation may or may not use as the IID of a local-only interface. [<82>](#)

The activation properties BLOB MUST contain properties marked Required in the following table, and MAY contain properties marked Optional.

Property Name	Section	Required or Optional
InstantiationInfoData	2.2.1.21.2.1	Required
ScmRequestInfoData	2.2.1.21.2.4	Required
LocationInfoData	2.2.1.21.2.6	Required
SecurityInfoData	2.2.1.21.2.7	Optional
ActivationContextInfoData	2.2.1.21.2.5	Optional
InstanceInfoData	2.2.1.21.2.3	Optional
SpecialPropertiesData	2.2.1.21.2.2	Optional

Servers SHOULD ignore properties that they do not recognize. [<83>](#)

ppActProperties: This MUST contain an **MInterfacePointer** that MUST contain an OBJREF_CUSTOM with a CLSID field set to CLSID_ActivationPropertiesOut (section [1.9](#)) and a **pObjectData** field that MUST contain an activation properties BLOB (section [2.2.1.21](#)). The **iid** field of the OBJREF portion of the structure MUST be set to a value that an implementation may or may not use as the IID of a local-only interface. [<84>](#)

The activation properties BLOB MUST contain all properties listed in the following table. Clients SHOULD ignore properties that they do not recognize.

Property Name	Section	Required or Optional
ScmReplyInfoData	2.2.1.21.2.8	Required
PropsOutInfo	2.2.1.21.2.9	Required

When processing this call, the object resolver MUST perform the operations specified in section [3.2.2.5.2.3](#).

3.2.2.5.2.3.3 IRemoteSCMAActivator::RemoteCreateInstance (Opnum 4)

The **RemoteCreateInstance (Opnum 4)** method is used by clients to create an object reference for the actual object.

```
HRESULT RemoteCreateInstance (
    [in] handle_t hRpc,
    [in] ORPCTHIS* orpcthis,
    [out] ORPCTHAT* orpcthat,
    [in,unique] MInterfacePointer* pUnkOuter,
    [in,unique] MInterfacePointer* pActProperties,
    [out] MInterfacePointer** ppActProperties
);
```

hRpc: This MUST specify an RPC binding handle as specified in [\[MS-RPCE\]](#) section 3.2.2.3.1 .

orpcthis: This MUST specify an ORPCTHIS. The **CONVERSION** field SHOULD contain the negotiated version as defined in section [1.7](#). The extensions field MUST be set to NULL.

orpcthat: This MUST contain an ORPCTHAT. The extensions field MUST be set to NULL.

pUnkOuter: This MUST be NULL, and MUST be ignored by the recipient.

pActProperties: This MUST specify an **MInterfacePointer** that MUST contain an OBJREF_CUSTOM with a CLSID field set to CLSID_ActivationPropertiesIn (see section [1.9](#)) and a **pObjectData** field that MUST contain an activation properties binary large object (BLOB) (see section [2.2.1.21](#)). The **iid** field of the **OBJREF** portion of the structure MUST be set to a value that an implementation may or may not use as the IID of a local-only interface. [<85>](#)

The activation properties BLOB MUST contain properties that are marked Required in the following table; and MAY contain properties that are marked Optional.

Property name	Section	Required or optional
InstantiationInfoData	2.2.1.21.2.1	Required
ScmRequestInfoData	2.2.1.21.2.4	Required
LocationInfoData	2.2.1.21.2.6	Required
SecurityInfoData	2.2.1.21.2.7	Optional
ActivationContextInfoData	2.2.1.21.2.5	Optional
InstanceInfoData	2.2.1.21.2.3	Optional
SpecialPropertiesData	2.2.1.21.2.2	Optional

Servers SHOULD ignore properties that they do not recognize. [<86>](#)

ppActProperties: This MUST contain an **MInterfacePointer** that MUST contain an OBJREF_CUSTOM with a CLSID field set to CLSID_ActivationPropertiesOut (see section [1.9](#)) and a **pObjectData** field that MUST contain an activation properties BLOB (see section [2.2.1.21](#)). The **iid** field of the OBJREF portion of the structure MUST be set to a value that an implementation may or may not use as the IID of a local-only interface. [<87>](#)

The activation properties BLOB MUST contain all properties listed in the following table. Clients SHOULD ignore properties that they do not recognize.

Property name	Section	Required or optional
ScmReplyInfoData	2.2.1.21.2.8	Required
PropsOutInfo	2.2.1.21.2.9	Required

When processing this call, the object resolver MUST perform the operations specified in section [3.2.2.5.2.3](#).

3.2.2.6 Timer Events

When the ping set timer expires, the object resolver SHOULD assume that the client is either dead or is unreachable and MUST perform the following:

- For each OID in the ping set entry:
 - It MUST look up the OID entry.
 - It MUST decrement the reference count of the OID entry held by the SETID entry.
 - If the reference count drops to zero:
 - It MUST remove the OID entry.
 - It MUST instruct the object exporter of the OID to reclaim the object.
- It MUST remove the ping set entry from the SETID table.

3.2.2.7 Other Local Events

Not applicable to object resolvers.

4 Protocol Examples

The following sections describe operations as used in common scenarios to illustrate the function of the DCOM Remote Protocol.

4.1 Object Activation + ORPC Call + Release Sequence

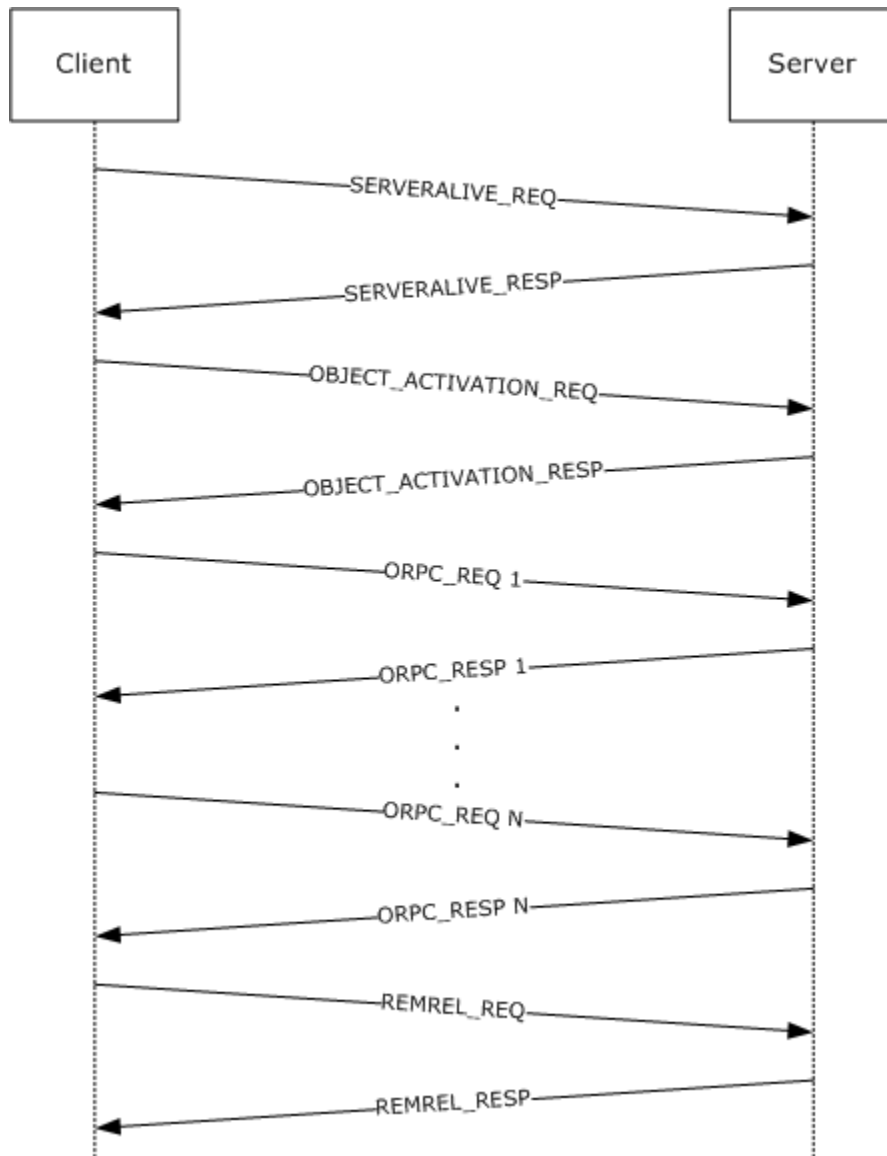


Figure 5: Object activation + ORPC call + release sequence

This example shows a sequence for a client requesting an object activation from a server, making a series of ORPC calls on the object, and then releasing the reference.

`SERVERALIVE_REQ`: Represents one or more RPC calls to the [IObjectExporter::ServerAlive](#) method or the [IObjectExporter::ServerAlive2](#) method to determine server aliveness and suitable

protocols. The client connects to the remote server name specified by the application or by a higher-layer protocol.

OBJECTACTIVATION_REQ: An RPC call to the object resolver service using the [IActivation](#) or the [IRemoteSCMAActivator](#) interface; the object resolver finds or creates an object exporter for the object.

ORPC_REQ: An ORPC call to the object exporter on the interface identified by the IPID.

REMREL_REQ: An ORPC call to the [IRemUnknown::RemRelease](#) method on the Remote Unknown of the object exporter containing the existing object reference.

4.2 QueryInterface + ORPC Call + Release Sequence

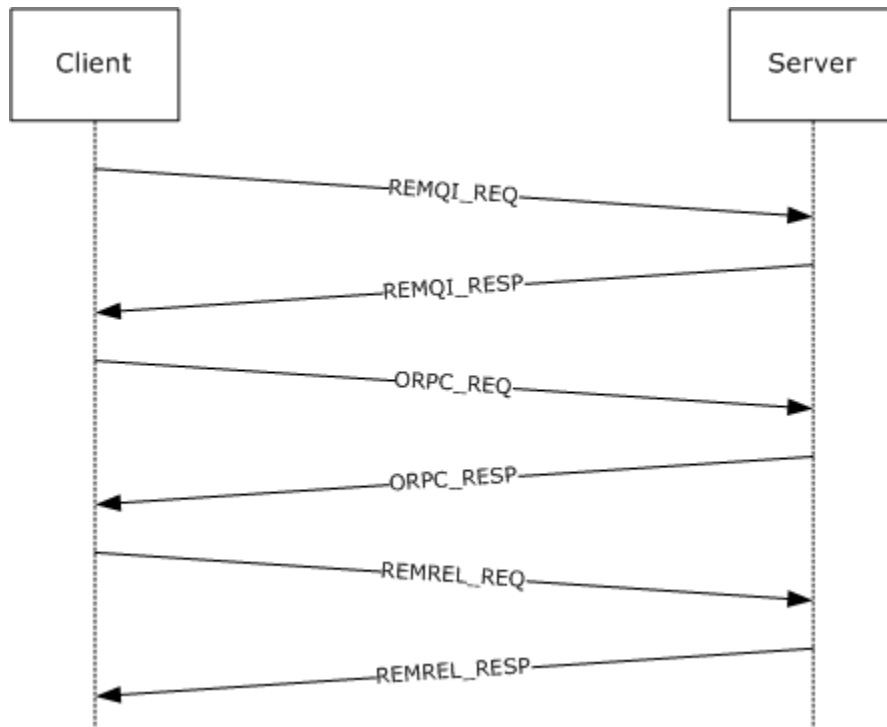


Figure 6: QueryInterface + ORPC call + release sequence

This example shows a sequence for a client requesting a new interface on an existing object reference, making an ORPC call on the new interface, and then releasing the interface.

REMQI_REQ: An ORPC call to the [IRemUnknown::RemQueryInterface](#) or [IRemUnknown2::RemQueryInterface2](#) method on the Remote Unknown of the object exporter containing the existing object reference.

ORPC_REQ: An ORPC call to the object exporter on the new interface identified by the IPID.

REMREL_REQ: An ORPC call to the [IRemUnknown::RemRelease](#) method of the object exporter containing the existing object reference.

4.3 Pinging Sequence

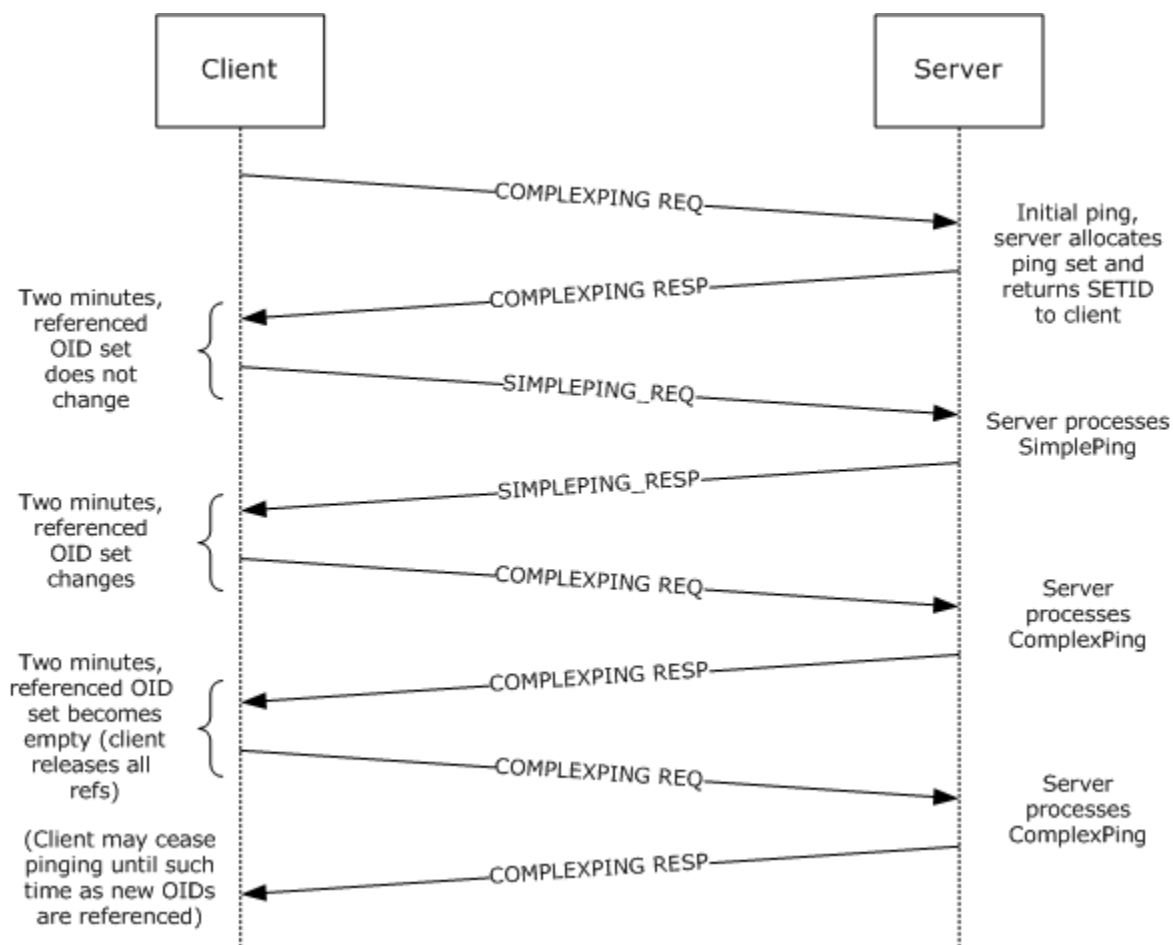


Figure 7: Client-to-server pinging sequence

This example shows a sequence for a client that pings the server to maintain object lifetimes.

COMPLEXPING_REQ: Represents one or more RPC calls to the [IObjectExporter::ComplexPing](#) method on the object resolver to allocate a new ping set or to add/remove one or more objects from an existing ping set.

SIMPLEPING_REQ: Represents one or more RPC calls to the [IObjectExporter::SimplePing](#) method on the object resolver to ping an existing ping set.

4.4 OXID Resolution Sequence

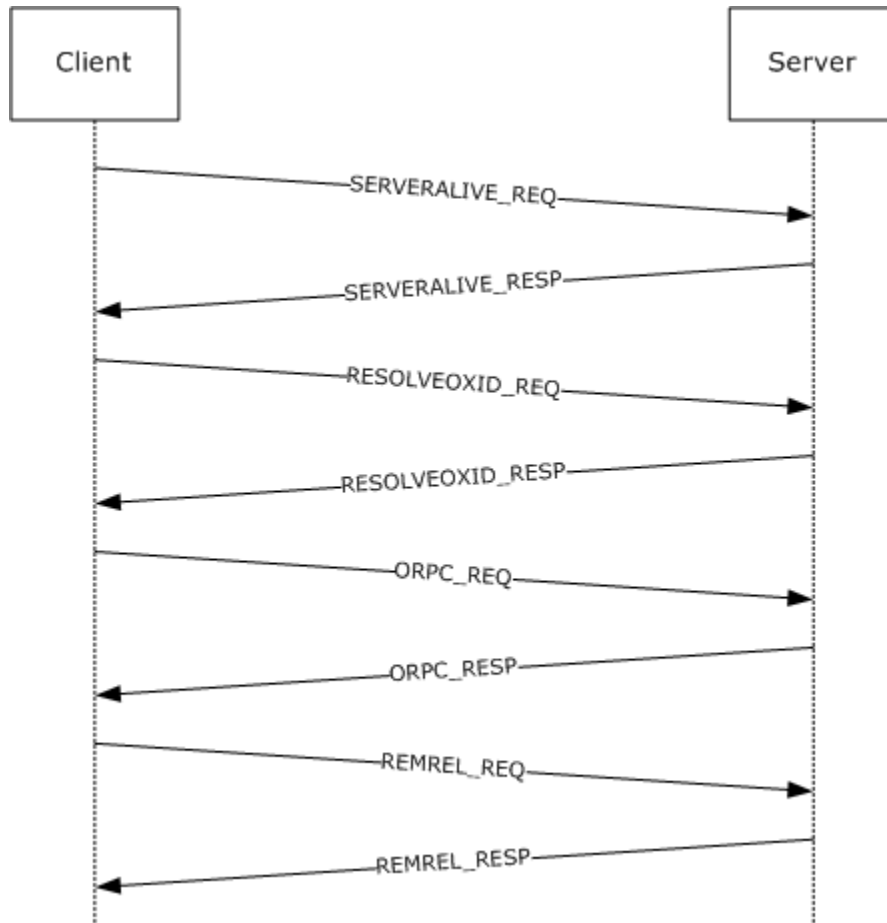


Figure 8: OXID resolution sequence

There is no requirement for the client to receive an OBJREF via one of the object activation interfaces: [IActivation](#) and [IRemoteSCMActivator](#). However, if the client receives an OBJREF via some other mechanism, the OBJREF does not contain sufficient information to make ORPC calls to the object exporter. In particular, the RPC endpoint information for the object exporter is obtainable only through the [IObjectExporter::ResolveOxid](#) method or the [IObjectExporter::ResolveOxid2](#) method.

This example shows a sequence for a client that has received an OBJREF via some undefined mechanism and needs to retrieve the RPC endpoint for the object exporter.

SERVERALIVE_REQ: Represents one or more RPC calls to the [IObjectExporter::ServerAlive](#) method or the [IObjectExporter::ServerAlive2](#) method to determine server aliveness and suitable protocols. The server addresses used by the client were contained within the saResAddr field of [OBJREF_STANDARD](#) contained in the OBJREF.

RESOLVEOXID_REQ: An RPC call to the **IObjectExporter::ResolveOxid** method or the **IObjectExporter::ResolveOxid2** method on the object resolver to retrieve the RPC binding information for the object exporter.

ORPC_REQ: An ORPC call to the object exporter on the interface identified by the IPID.

REMREL_REQ: An ORPC call to the [IRemUnknown::RemRelease](#) method on the Remote Unknown of the object exporter containing the existing object reference.

4.5 IDL Correlation Example for iid_is

iid_is is an IDL language extension that allows clients or servers to specify how an interface passed as a parameter is marshaled at run time, as opposed to being statically defined at compile time. For example:

```
HRESULT CreateInstance([in]REFIID riid,  
                      [out, iid_is(riid)]IUnknown** ppUnknown);
```

In this example, the client calls the method, passing the interface to be invoked via the *riid* parameter. When returning a successful result, the server must marshal the *ppUnknown* parameter as a [MInterfacePointer](#) containing an [OBJREF](#) instance with its **iid** field set to the value of *riid*.

5 Security

The following sections specify security considerations for administrators.

5.1 Security Considerations for Implementers

Implementations MAY decide to enforce security (as specified in [\[C706\]](#) section [2.7](#)) as needed on the processes and operations defined in this specification. [.<88>](#)

Implementers SHOULD review the security considerations as specified in [\[MS-RPCE\]](#) as these are also valid for the DCOM Remote Protocol and protocols based on the DCOM Remote Protocol.

5.2 Index of Security Parameters

None.

6 Appendix A: Full DCOM IDL

For ease of implementation, the full IDL is provided.

```
import "ms-dtyp.idl";

typedef GUID CLSID;
typedef GUID IID;
typedef unsigned hyper ID;
typedef ID OXID;
typedef ID OID;
typedef ID SETID;
typedef GUID IPID;
typedef GUID CID;

#define REFGUID const GUID *
#define REFIID const IID *

typedef REFGUID REFIPID;

typedef struct tagCOMVERSION
{
    unsigned short MajorVersion;
    unsigned short MinorVersion;
} COMVERSION;

typedef struct tagORPC_EXTENT
{
    GUID id;
    unsigned long size;
    [size_is((size+7)&~7)] byte data[];
} ORPC_EXTENT;

typedef struct tagORPC_EXTENT_ARRAY
{
    unsigned long size;
    unsigned long reserved;
    [size_is((size+1)&~1), unique] ORPC_EXTENT **extent;
} ORPC_EXTENT_ARRAY;

typedef struct tagORPCTHIS
{
    COMVERSION version;
    unsigned long flags;
    unsigned long reserved1;
    CID cid;
    [unique] ORPC_EXTENT_ARRAY *extensions;
} ORPCTHIS;

typedef struct tagORPCTHAT
{
    unsigned long flags;
    [unique] ORPC_EXTENT_ARRAY *extensions;
} ORPCTHAT;

typedef struct tagDUALSTRINGARRAY
```

```

{
    unsigned short wNumEntries;
    unsigned short wSecurityOffset;
    [size_is(wNumEntries)] unsigned short aStringArray[];
} DUALSTRINGARRAY;

enum tagCPFLAGS
{
    CPFLAG_PROPAGATE                = 0x1,
    CPFLAG_EXPOSE                   = 0x2,
    CPFLAG_ENVOY                    = 0x4,
};

typedef struct tagMInterfacePointer
{
    unsigned long                    ulCntData;
    [size_is(ulCntData)] byte abData[];
} MInterfacePointer;

typedef [unique] MInterfacePointer * PMInterfacePointer;

typedef struct tagErrorObjectData
{
    DWORD dwVersion;
    DWORD dwHelpContext;
    IID iid;
    [unique,string]wchar_t* pszSource;
    [unique,string]wchar_t* pszDescription;
    [unique,string]wchar_t* pszHelpFile;
} ErrorObjectData;

[
    uuid(4d9f4ab8-7d1c-11cf-861e-0020af6e7c57),
    pointer_default(unique)
]
interface IActivation
{
    const unsigned long MAX_REQUESTED_INTERFACES = 0x8000;
    const unsigned long MAX_REQUESTED_PROTSEQS = 0x8000;

error_status_t RemoteActivation(
    [in] handle_t                                hRpc,
    [in] ORPCTHIS                                *ORPCthis,
    [out] ORPCTHAT                                *ORPCthat,
    [in] GUID                                    *Clsid,
    [in, string, unique] wchar_t                  *pwszObjectName,
    [in, unique] MInterfacePointer                *pObjectStorage,
    [in] DWORD                                    ClientImpLevel,
    [in] DWORD                                    Mode,
    [in,range(1,MAX_REQUESTED_INTERFACES)]DWORD Interfaces,
    [in,unique,size_is(Interfaces)] IID            *pIIDs,
    [in,range(0,MAX_REQUESTED_PROTSEQS)]
        unsigned short                            cRequestedProtseqs,
    [in, size_is(cRequestedProtseqs)]
        unsigned short                            aRequestedProtseqs[],
    [out] OXID                                    *pOxid,
    [out] DUALSTRINGARRAY                        **ppdsaOxidBindings,
    [out] IPID                                    *pipidRemUnknown,

```



```

        [out] DWORD                                *pAuthnHint,
        [out] COMVERSION                            *pServerVersion,
        [out] HRESULT                              *phr,
        [out,size_is(Interfaces), disable_consistency_check]
            MInterfacePointer **ppInterfaceData,
        [out,size_is(Interfaces), disable_consistency_check]
            HRESULT                                *pResults
    );
}

[
    uuid(000001A0-0000-0000-C000-000000000046),
    pointer_default(unique)
]
interface IRemoteSCMAActivator
{
    HRESULT DummyQueryInterfaceIRemoteSCMAActivator(
        [in] handle_t rpc,
        [in] ORPCTHIS *orpcthis,
        [out] ORPCTHAT *orpcthat,
        [in] DWORD dummy );

    HRESULT DummyAddRefIRemoteSCMAActivator(
        [in] handle_t rpc,
        [in] ORPCTHIS *orpcthis,
        [out] ORPCTHAT *orpcthat,
        [in] DWORD dummy );

    HRESULT DummyReleaseIRemoteSCMAActivator(
        [in] handle_t rpc,
        [in] ORPCTHIS *orpcthis,
        [out] ORPCTHAT *orpcthat,
        [in] DWORD dummy );

    HRESULT RemoteGetClassObject(
        [in] handle_t rpc,
        [in] ORPCTHIS *orpcthis,
        [out] ORPCTHAT *orpcthat,
        [in,unique] MInterfacePointer *pActProperties,
        [out] MInterfacePointer **ppActProperties
    );

    HRESULT RemoteCreateInstance(
        [in] handle_t rpc,
        [in] ORPCTHIS *orpcthis,
        [out] ORPCTHAT *orpcthat,
        [in,unique] MInterfacePointer *pUnkOuter,
        [in,unique] MInterfacePointer *pActProperties,
        [out] MInterfacePointer **ppActProperties
    );
}

[
    uuid(99fcfec4-5260-101b-bbcb-00aa0021347a),
    pointer_default(unique)
]

interface IObjectExporter

```

```

{
    [idempotent] error_status_t ResolveOxid
    (
        [in]          handle_t          hRpc,
        [in]          OXID               *pOxid,
        [in]          unsigned short     cRequestedProtseqs,
        [in, ref, size_is(cRequestedProtseqs)]
            unsigned short     arRequestedProtseqs[],
        [out, ref] DUALSTRINGARRAY **ppdsaOxidBindings,
        [out, ref] IPID               *pipidRemUnknown,
        [out, ref] DWORD               *pAuthnHint
    );

    [idempotent] error_status_t SimplePing
    (
        [in] handle_t hRpc,
        [in] SETID    *pSetId
    );

    [idempotent] error_status_t ComplexPing
    (
        [in]          handle_t          hRpc,
        [in, out] SETID    *pSetId,
        [in]          unsigned short     SequenceNum,
        [in]          unsigned short     cAddToSet,
        [in]          unsigned short     cDelFromSet,
        [in, unique, size_is(cAddToSet)]  OID AddToSet[],
        [in, unique, size_is(cDelFromSet)]  OID DelFromSet[],
        [out]          unsigned short     *pPingBackoffFactor
    );

    [idempotent] error_status_t ServerAlive
    (
        [in]          handle_t          hRpc
    );

    [idempotent] error_status_t ResolveOxid2
    (
        [in]          handle_t          hRpc,
        [in]          OXID               *pOxid,
        [in]          unsigned short     cRequestedProtseqs,
        [in, ref, size_is(cRequestedProtseqs)]
            unsigned short     arRequestedProtseqs[],
        [out, ref] DUALSTRINGARRAY **ppdsaOxidBindings,
        [out, ref] IPID               *pipidRemUnknown,
        [out, ref] DWORD               *pAuthnHint,
        [out, ref] COMVERSION          *pComVersion
    );

    [idempotent] error_status_t ServerAlive2
    (
        [in]          handle_t          hRpc,
        [out, ref] COMVERSION          *pComVersion,
        [out, ref] DUALSTRINGARRAY **ppdsaOrBindings,
        [out, ref] DWORD               *pReserved
    );
}

```

```

typedef struct tagSTDOBJREF
{
    unsigned long    flags;
    unsigned long    cPublicRefs;
    OXID             oxid;
    OID              oid;
    IPID             ipid;
} STDOBJREF;

typedef struct tagREMQUIRESULT
{
    HRESULT          hResult;
    STDOBJREF        std;
} REMQUIRESULT;

typedef struct tagREMINTERFACEREF
{
    IPID             ipid;
    unsigned long    cPublicRefs;
    unsigned long    cPrivateRefs;
} REMINTERFACEREF;

typedef [disable_consistency_check] REMQUIRESULT* PREMQUIRESULT;
typedef [disable_consistency_check] MInterfacePointer*
        PMInterfacePointerInternal;

[
    object,
    uuid(00000000-0000-0000-C000-000000000046),
    pointer_default(unique)
]
interface IUnknown
{
    HRESULT Opnum0NotUsedOnWire(void);
    HRESULT Opnum1NotUsedOnWire(void);
    HRESULT Opnum2NotUsedOnWire(void);
};

[
    object,
    uuid(00000131-0000-0000-C000-000000000046),
    async_uuid(000e0131-0000-0000-C000-000000000046)
]
interface IRemUnknown : IUnknown
{
    HRESULT RemQueryInterface
    (
        [in] REFIID                      ripid,
        [in] unsigned long                cRefs,
        [in] unsigned short               cIids,
        [in, size_is(cIids)] IID          *iids,
        [out, size_is(,cIids)] PREMQUIRESULT *ppQUIResults
    );

    HRESULT RemAddRef
    (
        [in] unsigned short cInterfaceRefs,

```

```

        [in, size_is(cInterfaceRefs)] REMINTERFACEREF
            InterfaceRefs[],
        [out, size_is(cInterfaceRefs)] HRESULT *pResults
    );

HRESULT RemRelease
(
    [in] unsigned short cInterfaceRefs,
    [in, size_is(cInterfaceRefs)] REMINTERFACEREF
        InterfaceRefs[]
);
}

[
    object,
    uuid(00000143-0000-0000-C000-000000000046),
    async_uuid(000e0143-0000-0000-C000-000000000046)
]
interface IRemUnknown2 : IRemUnknown
{
    HRESULT RemQueryInterface2
    (
        [in] REFIPID                                ripid,
        [in] unsigned short                          cIids,
        [in, size_is(cIids)] IID                     *iids,
        [out, size_is(cIids)] HRESULT                 *phr,
        [out, size_is(cIids)] PMInterfacePointerInternal *ppMIF
    );
}

const unsigned long MIN_ACTPROP_LIMIT = 1;
const unsigned long MAX_ACTPROP_LIMIT = 10;
const unsigned long MIN_OPAQUE_BLOBS = 0;
const unsigned long MAX_OPAQUE_BLOBS = 10;
const unsigned long MAX_BLOBSIZE = 0x1000000;

typedef struct _COSERVERINFO
{
    DWORD dwReserved1;
    [string ] wchar_t* pwszName;
    DWORD * pdwReserved;
    DWORD dwReserved2;
} COSERVERINFO;

typedef struct _customREMOTE_REQUEST_SCM_INFO
{
    DWORD ClientImpLevel;
    [range (0, MAX_REQUESTED_PROTSEQS)] unsigned short
        cRequestedProtseqs;
    [size_is(cRequestedProtseqs)]
    unsigned short *pRequestedProtseqs;
} customREMOTE_REQUEST_SCM_INFO;

typedef struct _customREMOTE_REPLY_SCM_INFO
{
    OXID Oxid;
    DUALSTRINGARRAY *pdsaOxidBindings;
}

```

```

        IPID                                ipidRemUnknown;
        DWORD                                authnHint;
        COMVERSION                           serverVersion;
} customREMOTE_REPLY_SCM_INFO;

typedef struct tagCustomOpaqueData
{
    GUID guid;
    [range (8, MAX_BLOBSIZE)] unsigned long dataLength;
    unsigned long reserved1;
    unsigned long reserved2;
    [size_is(dataLength)] byte *data;
} CustomOpaqueData;

typedef struct tagInstantiationInfoData
{
    CLSID classId;
    DWORD classCtx;
    DWORD actvflags;
    long fIsSurrogate;
    [range (1,MAX_REQUESTED_INTERFACES)] DWORD cIID;
    DWORD instFlag;
    [size_is(cIID)] IID *pIID;
    DWORD thisSize;
    COMVERSION clientCOMVersion;
} InstantiationInfoData;

typedef struct tagLocationInfoData
{
    [string] wchar_t *machineName;
    DWORD processId;
    DWORD apartmentId;
    DWORD contextId;
} LocationInfoData;

typedef struct tagActivationContextInfoData
{
    long clientOK;
    long bReserved1;
    DWORD dwReserved1;
    DWORD dwReserved2;
    MInterfacePointer *pIFDClientCtx;
    MInterfacePointer *pIFDPrototypeCtx;
} ActivationContextInfoData;

typedef struct tagCustomHeader
{
    DWORD totalSize;
    DWORD headerSize;

    DWORD cOpaqueData;
    DWORD destCtx;
    [range (MIN_ACTPROP_LIMIT, MAX_ACTPROP_LIMIT)] DWORD cIifs;
    CLSID classInfoClsid;
    [size_is(cIifs)] CLSID *pclsid;
    [size_is(cIifs)] DWORD *pSizes;
    [size_is(cOpaqueData)] CustomOpaqueData *opaqueData;
} CustomHeader;

```

```

typedef struct tagPropsOutInfo
{
    [range (1, MAX_REQUESTED_INTERFACES)] DWORD cIifs;
    [size_is(cIifs)] IID *piid;
    [size_is(cIifs)] HRESULT *phresults;
    [size_is(cIifs)] MInterfacePointer **ppIntfData;
} PropsOutInfo;

typedef struct tagSecurityInfoData
{
    DWORD dwAuthnFlags;
    COSERVERINFO *pServerInfo;
    DWORD *pdwReserved;
} SecurityInfoData;

typedef struct tagScmRequestInfoData
{
    DWORD *pdwReserved;
    customREMOTE_REQUEST_SCM_INFO *remoteRequest;
} ScmRequestInfoData;

typedef struct tagScmReplyInfoData
{
    DWORD *pdwReserved;
    customREMOTE_REPLY_SCM_INFO *remoteReply;
} ScmReplyInfoData;

typedef struct tagInstanceInfoData
{
    [string] wchar_t *fileName;
    DWORD mode;
    MInterfacePointer *ifdROT;
    MInterfacePointer *ifdStg;
} InstanceInfoData;

typedef enum
{
    SPD_FLAG_USE_CONSOLE_SESSION = 0x00000001,
    SPD_FLAG_USE_DEFAULT_AUTHN_LVL = 0x00000002,
} SPD_FLAGS;

typedef struct tagSpecialPropertiesData
{
    unsigned long dwSessionId;
    long fRemoteThisSessionId;
    long fClientImpersonating;
    long fPartitionIDPresent;
    DWORD dwDefaultAuthnLvl;
    GUID guidPartition;
    DWORD dwPRTFlags;
    DWORD dwOrigClsctx;
    DWORD dwFlags;
    DWORD dwPid;
    unsigned __int64 hwnd;
    DWORD dwReserved[5];
} SpecialPropertiesData;

```

[MS-DCOM] – v20080207

Distributed Component Object Model (DCOM) Remote Protocol Specification

Copyright © 2008 Microsoft Corporation.

Release: Thursday, February 7, 2008

7 Appendix B: Windows Behavior

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

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

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

[<1> Section 1.8:](#) Windows only uses the values in [\[MS-ERREF\]](#) section 2.2.

[<2> Section 1.9:](#) Windows object resolver services always use the well-known endpoints specified in [\[MS-RPCE\]](#) section 2.1, and will never register their interfaces with the RPC endpoint mapper. Windows DCOM clients correctly interoperate with a server whose object resolver service registers its interfaces with the RPC endpoint mapper.

[<3> Section 2.1:](#) Windows DCOM servers register the following security providers:

- RPC_C_AUTHN_GSS_NEGOTIATE
- RPC_C_AUTHN_WINNT
- RPC_C_AUTHN_GSS_KERBEROS

[<4> Section 2.2.1.11:](#) The DCOM versions supported by different platforms are:

	Windows 95/98	Windows NT	Windows 2000	Windows XP	Windows Server 2003	Windows Vista
DCOM version supported	5.4	5.4	5.6	5.7	5.7	5.7

[<5> Section 2.2.1.11:](#) Windows uses version 5.7, not to indicate any change in DCOM, but rather in the marshaling of the UDT type specified in [\[MS-OAUT\]](#) section 2.2.25.1.

[<6> Section 2.2.1.13.1:](#) This specification defines two formats for the [ORPC_EXTENT](#) structure. See section [2.2.1.20](#).

[<7> Section 2.2.1.17.2:](#) Windows will not perform garbage collection pinging for objects unmarshaled with SORF_NOPING.

[<8> Section 2.2.1.17.6:](#) Windows treats this field as the CLSID for an object that both implements the IMarshal interface and is capable of unmarshaling the **pObjectData** field. For more information, see [\[MSDN-IMarshal\]](#).

<9> [Section 2.2.1.18.3](#): Windows supports a subset of the constants. For details, see section [3.2.2.3](#).

<10> [Section 2.2.1.19](#): Windows DCOM clients set the **iid** field to {000001C0-0000-0000-C000-000000000046}.

<11> [Section 2.2.1.19](#): Windows DCOM clients set the **iid** field to {000001C0-0000-0000-C000-000000000046}.

<12> [Section 2.2.1.19](#): Windows DCOM servers reply with the error `RPC_E_INVALID_OBJREF` ([\[MS-ERREF\]](#) section 2.1) if the [Context](#) structure has a nonzero **dwNumExtents**.

<13> [Section 2.2.1.19](#): Windows DCOM servers reply with the error `RPC_E_INVALID_OBJREF` ([\[MS-ERREF\]](#) section 2.1) if the [Context](#) structure has a nonzero **cbNumExtents**.

<14> [Section 2.2.1.19](#): Windows DCOM clients set this field to a value from the `MSHLFLAGS` enumeration. For more information, see [\[MSDN-MSHLFLAGS\]](#).

<15> [Section 2.2.1.20.1](#): Windows DCOM clients and servers process the [OBJREF](#) supplied in the data field of this ORPC extension as a reference to an object that supports the `IErrorInfo` interface. For more information, see [\[MSDN-IERRORINFO\]](#).

<16> [Section 2.2.1.20.2](#): Optionally specifies the index for a help topic in the help file specified by the **HelpFile** field.

<17> [Section 2.2.1.20.2](#): Optionally specifies a human-readable string containing the name of the component returning the error.

<18> [Section 2.2.1.20.2](#): Optionally specifies a human-readable string containing a description of the error.

<19> [Section 2.2.1.20.2](#): Optionally specifies a path to a Windows Help file containing a Help topic that provides further information for the error.

<20> [Section 2.2.1.20.4](#): Windows DCOM clients set this value to the size (in bytes) of the body of the RPC PDU containing this structure.

<21> [Section 2.2.1.20.4](#): This field is used by applications or higher-layer protocols. Windows DCOM clients and servers ignore this field.

<22> [Section 2.2.1.21.1](#): Windows DCOM clients set this field to `MSHCTX_DIFFERENTMACHINE` (0x00000002), which is a value from the `MSHCTX` enumeration. For more information, see [\[MSDN-MSHCTX\]](#).

<23> [Section 2.2.1.21.2.1](#): Windows DCOM clients set this field to one or more values from the `CLSCTX` enumeration. For more information, see [\[MSDN-CLSCTX\]](#).

<24> [Section 2.2.1.21.2.2](#): Windows DCOM clients set this field to the Windows Terminal Server session ID (for more information, see [\[MSDN-WSI\]](#)) that the client is requesting the object exporter for the activation to run within, or to 0xFFFFFFFF to indicate no preference.

<25> [Section 2.2.1.21.2.2](#): Windows DCOM clients set this field to `TRUE` if the client was impersonating when the activation request was originated, and to `FALSE` otherwise. Windows DCOM servers ignore this field. For more information, see [\[MSDN-CI\]](#).

<26> [Section 2.2.1.21.2.2](#): Windows DCOM clients set this field to `FALSE` (0x00000000) if **guidPartition** is not set, and to `TRUE` (0x00000001) otherwise. Windows DCOM servers use the **guidPartition** field if **fPartitionIDPresent** is set to `TRUE`.

<27> [Section 2.2.1.21.2.2:](#) Windows DCOM clients set this field to an RPC authentication constant (see section [2.2.1.26.1](#)).

<28> [Section 2.2.1.21.2.2:](#) The value contains a GUID used by applications or higher-layer protocols.

<29> [Section 2.2.1.21.2.2:](#) Windows DCOM clients set this field to the unmodified CLSCTX value specified by the client when the activation request was originated. For more information, see [\[MSDN-CLSCTX\]](#).

<30> [Section 2.2.1.21.2.2:](#) Windows Servers treat the **dwFlags** field as follows:

Note Object exporters MUST treat any value other than 1 as if the client specified a value of zero.

Value	Meaning
0x00000000	The object exporter creates the new object in the Windows Terminal Server session specified in the dwSessionID field. The object exporter returns CO_E_RUNAS_LOGON_FAILURE (as specified in [MS-ERREF] section 2.1) if this requirement cannot be met.
0x00000001	The object exporter creates the new object in the currently-active Windows Terminal Server console session. The object exporter returns CO_E_RUNAS_LOGON_FAILURE (as specified in [MS-ERREF] section 2.1) if this requirement cannot be met.

<31> [Section 2.2.1.21.2.3:](#) Windows DCOM clients set this to a file name passed to the CoGetInstanceFromFile API. For more information, see [\[MSDN-CoGetInstanceFromFile\]](#).

<32> [Section 2.2.1.21.2.3:](#) Windows DCOM clients set this field to the unmodified STGM constant. For more information, see [\[MSDN-STGMC\]](#).

<33> [Section 2.2.1.21.2.3:](#) Windows DCOM clients set this to the IStorage reference passed to the CoGetInstanceFromIStorage API. For more information, see [\[MSDN-CoGetInstanceFromIStorage\]](#).

<34> [Section 2.2.1.21.2.4.1:](#) Windows DCOM clients set this field to the value 2.

<35> [Section 2.2.1.21.2.5:](#) Applications or higher-layer protocols use this field.

<36> [Section 2.2.1.21.2.7:](#) Windows DCOM clients include this structure; Windows DCOM servers ignore it.

<37> [Section 2.2.1.21.2.7:](#) Windows DCOM clients send a [COSERVERINFO](#) structure in this field as specified. Windows DCOM servers ignore this field.

<38> [Section 2.2.1.21.2.7.1:](#) Windows DCOM clients set **pwszName** to the remote server name specified by the client when requesting the activation. Windows DCOM servers ignore this field.

<39> [Section 2.2.1.21.2.8.1:](#) Windows DCOM servers return an RPC authentication level that denotes the minimum authentication level at which the object exporter can be called. Windows DCOM clients make calls to object exporters at an authentication level that is at least as high as the **authnHint** returned from the object server.

<40> [Section 3:](#) All Windows implementations support both roles simultaneously.

<41> [Section 3.1:](#) For details on which versions of Windows support which version of the DCOM Remote Protocol, see section [2.2.1.11](#).

<42> [Section 3.1.4.1.1.2:](#) Windows DCOM clients prior to Windows XP SP2 and Windows Server 2003 SP1 specify `RPC_C_AUTHN_LEVEL_CONNECT` (see [\[MS-RPCE\]](#) section 2.2.1.1.8) as the default authentication level value for the call.

All other Windows DCOM clients specify the higher of the `LegacyAuthenticationLevel` value (for more information, see [\[MSDN-LegacyAuthenticationLevel\]](#)) and `RPC_C_AUTHN_LEVEL_CONNECT` (see [\[MS-RPCE\]](#) section 2.2.1.1.8) as the default authentication level value for the call.

<43> [Section 3.1.4.1.1.2:](#) Windows DCOM clients specify `RPC_C_IMPL_LEVEL_IMPERSONATE` (see [\[MS-RPCE\]](#) section 2.2.1.1.9) as the default impersonation level value for the call.

<44> [Section 3.1.4.1.2:](#) Windows clients will acquire an object reference for the IID specified by the application.

<45> [Section 3.1.4.1.2.2:](#) Windows DCOM clients specify `RPC_C_AUTHN_LEVEL_CONNECT` (see [\[MS-RPCE\]](#) section 2.2.1.1.8) as the authentication level for the call.

<46> [Section 3.1.4.1.2.2:](#) Windows DCOM clients specify `RPC_C_IMPL_LEVEL_IDENTIFY` (see [\[MS-RPCE\]](#) section 2.2.1.1.9) as the impersonation level for the call.

<47> [Section 3.1.4.2:](#) Windows DCOM clients use the `LegacyAuthenticationLevel` value (see [\[MSDN-LegacyAuthenticationLevel\]](#) for more information) as the client's authentication level value.

<48> [Section 3.1.4.2:](#) Windows DCOM clients use the `LegacyImpersonationLevel` value (see [\[MSDN-LegacyImpersonationLevel\]](#) for more information) as the default impersonation level value.

<49> [Section 3.1.4.2:](#) Windows DCOM clients specify the **extensions** field if well-known [ORPC Extensions](#) (section 2.2.1.20) are supplied by the application.

<50> [Section 3.1.4.2:](#) Windows 2000, Windows XP, Windows XP SP1, Windows XP SP2, Windows Server 2003 and Windows Server 2003 SP1 DCOM clients optionally append extra data to the end of an ORPC request. The extra data, if present, must be ignored by the recipient.

Windows Server 2003 SP2, Windows Vista and Windows Server 2008 DCOM clients do not append extra data.

<51> [Section 3.1.4.2:](#) Windows DCOM clients return the extensions field to the application if well-known [ORPC Extensions](#) are present in the [ORPCTHAT](#) structure.

<52> [Section 3.1.4.4.1:](#) Windows DCOM clients use private references when the secure reference counting feature is enabled in the DCOM application using the `EOAC_SECURE_REFS` capability. For more information, see [\[MSDN-EOLE AUTHENTICATION CAPABILITIES\]](#).

<53> [Section 3.1.4.4.2:](#) Windows DCOM clients use private reference counts when the secure reference counting feature is enabled using the `EOAC_SECURE_REFS` capability. For more information, see [\[MSDN-EOLE AUTHENTICATION CAPABILITIES\]](#).

<54> [Section 3.1.6.1:](#) Windows DCOM clients specify `RPC_C_AUTHN_LEVEL_CONNECT` (see [\[MS-RPCE\]](#) section 2.2.1.1.8) as the authentication level for the call.

<55> [Section 3.1.6.1:](#) Windows DCOM clients specify `RPC_C_IMPL_LEVEL_IDENTIFY` (see [\[MS-RPCE\]](#) section 2.2.1.1.9) as the impersonation level for the call.

<56> [Section 3.2.1.5.1:](#) Windows servers will set the `SORF_NOPING` flag if the application specifies the `MSHLFLAGS_NOPING` flag in the `mshlflags` parameter to the `CoMarshalInterface` API. For more information, see [\[MSDN-CoMarshalInterface\]](#).

<57> [Section 3.2.1.5.4:](#) Windows DCOM servers use the LegacyAuthenticationLevel value (see [MSDN-LegacyAuthenticationLevel] for more information) as the object exporter's default authentication level setting.

<58> [Section 3.2.1.5.4:](#) Windows DCOM servers prior to Windows XP SP2 and Windows Server 2003 SP1 use the DefaultAccessPermission (see [MSDN-DefaultAccessPermission] for more information) or the AccessPermission of the object exporter (see [MSDN-AccessPermission] for more information) as the default value of the permissions.

For all other Windows DCOM servers, the default value of the permissions consists of both:

- The MachineAccessRestriction (see [MSDN-MachineAccessRestriction] for more information).
- The DefaultAccessPermission (see [MSDN-DefaultAccessPermission] for more information) or the AccessPermission that is specific to the object exporter (see [MSDN-AccessPermission] for more information).

<59> [Section 3.2.1.5.4:](#) Windows object exporters use an application-specified message filter. For more information, see [\[MSDN-IMessageFilter\]](#).

<60> [Section 3.2.1.5.4:](#) Windows DCOM server object exporters supply the well-known ORPC extensions (see section [2.2.1.20](#)), if present, to applications and higher-layer protocols.

<61> [Section 3.2.1.5.4:](#) Windows DCOM server object exporters return the **extensions** field supplied by the well-known ORPC extensions (see section [2.2.1.20](#)), if present.

<62> [Section 3.2.1.5.4:](#) Windows 2000, Windows XP, Windows XP SP1, Windows XP SP2, Windows Server 2003 and Windows Server 2003 SP1 DCOM servers optionally append extra data to the end of an ORPC response. The extra data, if present, must be ignored by the recipient.

Windows Server 2003 SP2, Windows Vista and Windows Server 2008 DCOM servers do not append extra data.

<63> [Section 3.2.1.5.6.1.2:](#) Windows DCOM server object exporters require security on a RemAddRef call that specifies private reference counts. They will associate the private reference counts with the security identity of the client that makes the RemAddRef call.

<64> [Section 3.2.1.5.6.1.3:](#) Windows DCOM server object exporters require security on a RemRelease call that specifies private reference counts. They will verify that the security identity of the client that makes the RemRelease call has previously allocated at least that many private reference counts in the IPID entry.

<65> [Section 3.2.1.5.8:](#) Opnums reserved for local use apply to Windows as follows.

opnum	Description
0-2	Just returns RPC_E_INVALID_HEADER. It is never used.

Windows clients internally map the three IUnknown interface methods to the three methods of the [IRemUnknown](#) interface.

<66> [Section 3.2.2.3:](#) By default, Windows object resolvers listen by way of the following RPC protocols:

	Windows NT 4.0	Windows 2000	Windows XP	Windows Server 2003	Windows Vista
ncacn_ip_tcp	X	X	X	X	X
ncacn_spx	X	X	X		
ncacn_nb_nb	X	X	X		
ncacn_nb_ipx	X	X	X		
Ncadg_ip_udp	X				
Ncadg_ipx	X				

<67> [Section 3.2.2.5.1.1:](#) Windows DCOM servers return the minimum accepted authentication level of the object exporter in this field. Windows DCOM clients by default make calls to the object exporter, at least at this level of authentication.

<68> [Section 3.2.2.5.1.1:](#) Windows DCOM servers prior to Windows XP SP2 and Windows Server 2003 SP1 do not check permissions when processing this call.

All other Windows DCOM servers check permissions when processing this call. They use the MachineAccessRestriction (see [MSDN-MachineAccessRestriction] for more information) as the default value of the permissions.

<69> [Section 3.2.2.5.1.2:](#) Windows DCOM servers prior to Windows XP SP2 and Windows Server 2003 SP1 do not check permissions when processing this call.

All other Windows DCOM servers check permissions when processing this call. They use the MachineAccessRestriction (see [MSDN-MachineAccessRestriction] for more information) as the default value of the permissions.

<70> [Section 3.2.2.5.1.3:](#) Windows DCOM servers return a PingBackoffFactor of zero; Windows DCOM clients ignore any value returned by the server.

<71> [Section 3.2.2.5.1.3:](#) Windows DCOM servers prior to Windows XP SP2 and Windows Server 2003 SP1 do not check permissions when processing this call.

All other Windows DCOM servers check permissions when processing this call. They use the MachineAccessRestriction (see [MSDN-MachineAccessRestriction] for more information) as the default value of the permissions.

<72> [Section 3.2.2.5.1.5:](#) Windows DCOM servers return the minimum accepted authentication level of the object exporter in this field. Windows DCOM clients by default make calls to the object exporter, at least at this level of authentication.

<73> [Section 3.2.2.5.2.2.1:](#) Windows DCOM clients never call this method; Windows DCOM servers return E_NOTIMPL (as specified in [\[MS-ERREF\]](#) section 2.1) if this method is called.

<74> [Section 3.2.2.5.2.2.2](#): Windows DCOM clients never call this method; Windows DCOM servers return E_NOTIMPL (as specified in [\[MS-ERREF\]](#) section 2.1) if this method is called.

<75> [Section 3.2.2.5.2.2.3](#): Windows DCOM clients never call this method; Windows DCOM servers return E_NOTIMPL (as specified in [\[MS-ERREF\]](#) section 2.1) if this method is called.

<76> [Section 3.2.2.5.2.3](#): All server versions of the DCOM protocol prior to Windows XP SP2 and Windows Server 2003 SP1 use the DefaultLaunchPermission (see [\[MSDN-DefaultLaunchPermission\]](#) for more information) or the LaunchPermission that is specific to the object exporter (see [\[MSDN-LaunchPermission\]](#) for more information) as the default value of the permissions.

For all other server versions of the DCOM protocol, the default value of the permissions consists of the following:

- The MachineAccessRestriction (see [\[MSDN-MachineAccessRestriction\]](#) for more information).
- The MachineLaunchRestriction (see [\[MSDN-MachineLaunchRestriction\]](#) for more information).
- The DefaultLaunchPermission (see [\[MSDN-DefaultLaunchPermission\]](#) for more information) or the LaunchPermission that is specific to the object exporter (see [\[MSDN-LaunchPermission\]](#) for more information).

<77> [Section 3.2.2.5.2.3.1](#): Windows DCOM clients set this to a file name passed to the CoGetInstanceFromFile API. For more information, see [\[MSDN-CoGetInstanceFromFile\]](#).

<78> [Section 3.2.2.5.2.3.1](#): Windows DCOM clients set this to the IStorage reference passed to the CoGetInstanceFromIStorage API. For more information, see [\[MSDN-CoGetInstanceFromIStorage\]](#).

<79> [Section 3.2.2.5.2.3.1](#): Windows DCOM clients set this field to the value 2.

<80> [Section 3.2.2.5.2.3.1](#): If the DCOM application passes a file name to the CoGetInstanceFromFile API. For more information, see [\[MSDN-CoGetInstanceFromFile\]](#).

<81> [Section 3.2.2.5.2.3.1](#): Windows DCOM servers return the minimum accepted authentication level of the object exporter in this field. Windows DCOM clients by default make calls to the object exporter at least at this level of authentication.

<82> [Section 3.2.2.5.2.3.2](#): Windows DCOM clients set the **iid** field to {000001A2-0000-0000-C000-000000000046}.

<83> [Section 3.2.2.5.2.3.2](#): Windows DCOM clients send all the properties (including optional properties) listed in the following table, except InstanceInfoData. InstanceInfoData is sent only when the DCOM application makes a persistent activation request.

Property Name	Section	Required or Optional
InstantiationInfoData	2.2.1.21.2.1	Required
ScmRequestInfoData	2.2.1.21.2.4	Required
LocationInfoData	2.2.1.21.2.6	Required
SecurityInfoData	2.2.1.21.2.7	Optional
ActivationContextInfoData	2.2.1.21.2.5	Optional
InstanceInfoData	2.2.1.21.2.3	Optional

Property Name	Section	Required or Optional
SpecialPropertiesData	2.2.1.21.2.2	Optional

<84> [Section 3.2.2.5.2.3.2:](#) Windows DCOM servers set the **iid** field to {000001A3-0000-0000-C000-000000000046}.

<85> [Section 3.2.2.5.2.3.3:](#) Windows DCOM clients set the **iid** field to {000001A2-0000-0000-C000-000000000046}.

<86> [Section 3.2.2.5.2.3.3:](#) Windows DCOM clients send all the properties (including Optional properties) listed in the following table, except InstanceInfoData. InstanceInfoData is sent only when the DCOM application makes a persistent activation request.

Property name	Section	Required or optional
InstantiationInfoData	2.2.1.21.2.1	Required
ScmRequestInfoData	2.2.1.21.2.4	Required
LocationInfoData	2.2.1.21.2.6	Required
SecurityInfoData	2.2.1.21.2.7	Optional
ActivationContextInfoData	2.2.1.21.2.5	Optional
InstanceInfoData	2.2.1.21.2.3	Optional
SpecialPropertiesData	2.2.1.21.2.2	Optional

<87> [Section 3.2.2.5.2.3.3:](#) Windows DCOM servers set the **iid** field to {000001A3-0000-0000-C000-000000000046}.

<88> [Section 5.1:](#) If the application enables the EOAC_SECURE_REFS capability. For more information, see [\[MSDN-EOLE AUTHENTICATION CAPABILITIES\]](#). The default Windows security configuration requires the client to specify security on the activation requests and ORPC requests.

8 Index

A

Abstract data model

[client](#)
[object exporter](#)
[object resolver](#)

[Acquiring additional interfaces on the object](#)

[Activation](#)

[Activation - higher-layer triggered events](#)

[Activation properties](#)

[Activation Properties Blob packet](#)

[ActivationContextInfoData structure](#)

[Allocating OID entries](#)

[Allocating OXID entries](#)

[Applicability](#)

C

[Capability negotiation](#)

[Causality identifiers](#)

Client

[abstract data model](#)
[higher-layer triggered events](#)
[initialization](#)
[local events](#)
[message processing](#)
[overview](#)
[sequencing rules](#)
[timer events](#)
[timers](#)

[ComplexPing method](#)

[COMVERSION structure](#)

[Constant definitions](#)

[Context ORPC Extension packet](#)

[Context packet](#)

[COSERVERINFO structure](#)

[CustomHeader structure](#)

[customREMOTE_REPLY_SCM_INFO structure](#)

[customREMOTE_REQUEST_SCM_INFO structure](#)

D

Data model - abstract

[client](#)
[object exporter](#)
[object resolver](#)

[Data types](#)

[DATAELEMENT packet](#)

[DCOM IDL](#)

[Deleting OID entries](#)

[Determining RPC binding information](#)

[DUALSTRINGARRAY](#)

[DUALSTRINGARRAY packet](#)

[DUALSTRINGARRAY structure](#)

[DummyAddRefIRemoteSCMAActivator](#)

E

[EntryHeader packet](#)

[Error information ORPC extension](#)

[error_status_t](#)

[ErrorInfoString packet](#)

[ErrorObjectData packet](#)

Examples

[IDL correlation for iid is example](#)

[object activation + ORPC call + release sequence](#)

[example](#)

[overview](#)

[OXID resolution sequence example](#)

[pinging sequence example](#)

[QueryInterface + ORPC call + release sequence](#)

[example](#)

F

[Fields - vendor-extensible](#)

[Full DCOM IDL](#)

G

[Glossary](#)

[GUID](#)

H

Higher-layer triggered events

[client](#)
[object exporter](#)
[object resolver](#)

[HRESULT](#)

I

[IActivation methods](#)

[IDL - DCOM](#)

[IDL correlation for iid is example](#)

[iid is IDL Attribute](#)

[Implementer - security considerations](#)

[Index of security parameters](#)

[Informative references](#)

Initialization

[client](#)
[object exporter](#)
[object resolver](#)

[InstanceInfoData structure](#)

[InstantiationInfoData structure](#)

[Introduction](#)

[IRemoteSCMAActivator methods](#)

[IRemoteSCMAActivator::](#)

[DummyAddRefIRemoteSCMAActivator](#)

[IRemUnknown methods](#)

[IRemUnknown2 methods](#)

[Issuing activation request](#)

L

[Lazy protocol registration](#)

Local events

[client](#)
[Object exporter](#)
[object resolver](#)
[LocationInfoData structure](#)

M

[Marshaling object references](#)
Message processing
 [client](#)
 [object exporter](#)
 [object resolver](#)
Messages
 [data types](#)
 [overview](#)
 [syntax](#)
 [transport](#)
[MInterfacePointer structure](#)

N

[Normative references](#)

O

[Object activation + ORPC call + release sequence example](#)
[Object exporter](#)
 [abstract data model](#)
 [higher-layer triggered events](#)
 [initialization](#)
 [local events](#)
 [message processing](#)
 [overview](#)
 [sequencing rules](#)
 [timer events](#)
 [timers](#)
[object IDL Attribute](#)
[Object lifetime](#)
[Object marshaling](#)
[Object reclamation](#)
[Object reference - unmarshaling](#)
[Object reference marshaling](#)
[Object reference unmarshaling](#)
Object references ([section 1.3.2](#), [section 3.1.4.1](#), [section 3.1.4.3](#))
Object resolver
 [abstract data model](#)
 [higher-layer triggered events](#)
 [initialization](#)
 [local events](#)
 [message processing](#)
 [overview](#)
 [sequencing rules](#)
 [timer events](#)
 [timers](#)
[Object resolver service](#)
[OBJREF packet](#)
[OBJREF_CUSTOM packet](#)
[OBJREF_EXTENDED packet](#)
[OBJREF_HANDLER packet](#)

[OBJREF_STANDARD packet](#)
OID entries
 [allocating](#)
 [deleting](#)
[OID/IPID/resolver table - updating after unmarshaling](#)
[ORPC calls](#)
[ORPC extensions](#)
ORPC invocations ([section 3.1.4.2](#), [section 3.2.1.5.4](#))
[ORPC_EXTENT structure](#)
[ORPC_EXTENT_ARRAY structure](#)
[ORPCTHAT](#)
[ORPCTHAT structure](#)
[ORPCTHIS](#)
[ORPCTHIS structure](#)
[Overview \(synopsis\)](#)
[OXID entries - allocating](#)
OXID resolution
 [issuing resolution request](#)
 [RPC binding information](#)
[OXID resolution sequence example](#)
[OXID table - client - updating](#)
[OXID table - updating after unmarshaling](#)

P

[Parameters - security index](#)
Pinging ([section 3.1.6.1](#), [section 3.2.1.6.1](#))
[Pinging sequence example](#)
[Preconditions](#)
[Prerequisites](#)
[PROPMARSHALHEADER packet](#)
[PropsOutInfo structure](#)

Q

[QueryInterface + ORPC call + release sequence example](#)

R

[Reference counts](#)
References
 [informative](#)
 [normative](#)
 [overview](#)
[Relationship to other protocols](#)
[Releasing reference counts on an interface](#)
[RemAddRef method](#)
[REMINTERFACEREF structure](#)
[RemoteActivation method](#)
[RemoteCreateInstance method](#)
[RemoteGetClassObject method](#)
[REMQIRERESULT structure](#)
[RemQueryInterface method](#)
[RemQueryInterface2 method](#)
[RemRelease method](#)
[Requesting reference counts on an interface](#)
[ResolveOxid method](#)
[ResolveOxid2 method](#)
[RIDL range constants](#)
[RPC Authentication level constants](#)

[RPC Authentication service constants](#)
[RPC binding information - OXID resolution](#)

W

[Windows behavior](#)

S

[ScmReplyInfoData structure](#)
[ScmRequestInfoData structure](#)

Security

[implementer considerations](#)
[overview](#)
[parameter index](#)
[SECURITYBINDING packet](#)
[SecurityInfoData structure](#)

Sequencing rules

[client](#)
[object exporter](#)
[object resolver](#)
[ServerAlive method](#)
[ServerAlive2 method](#)
[SimplePing method](#)
[SpecialPropertiesData structure](#)
[Standards assignments](#)
[STDOBJREF](#)
[STDOBJREF packet](#)
[STDOBJREF structure](#)
[STRINGBINDING packet](#)

Syntax

[data types](#)
[overview](#)

T

Timer events

[client](#)
[object exporter](#)
[object resolver](#)

Timers

[client](#)
[object exporter](#)
[object resolver](#)

Transport

Triggered events - higher-layer

[client](#)
[object exporter](#)
[object resolver](#)

U

[Unmarshaling an object reference](#)
[Updating client OXID table - activation](#)

Updating client tables after unmarshaling

[OID/IPID/Resolver tables](#)
[overview](#)
[OXID table](#)

V

[Vendor-extensible fields](#)
[Versioning](#)