

[MS-DMRP]: Disk Management 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
05/11/2007	0.1		MCPPE Milestone 4 Initial Availability
08/10/2007	1.0	Major	Updated and revised the technical content.
09/28/2007	1.0.1	Editorial	Revised and edited the technical content.
10/23/2007	1.1	Minor	Updated the IDL.
11/30/2007	2.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
01/25/2008	2.0.1	Editorial	Revised and edited the technical content.

Table of Contents

1	Introduction	8
1.1	Glossary	8
1.2	References	10
1.2.1	Normative References	10
1.2.2	Informative References.....	10
1.3	Protocol Overview (Synopsis).....	10
1.4	Relationship to Other Protocols.....	11
1.5	Prerequisites/Preconditions	11
1.6	Applicability Statement	11
1.7	Versioning and Capability Negotiation.....	12
1.8	Vendor-Extensible Fields	12
1.9	Standards Assignments.....	12
2	Messages	13
2.1	Transport	13
2.2	Common Data Types	13
2.2.1	HRESULT Return Codes	13
2.2.2	MAX_FS_NAME_SIZE Constant	27
2.2.3	REGIONTYPE	27
2.2.4	VOLUMETYPE	28
2.2.5	VOLUMELAYOUT	28
2.2.6	REQSTATUS	29
2.2.7	REGIONSTATUS.....	29
2.2.8	VOLUMESTATUS	30
2.2.9	LdmObjectId	30
2.2.10	VOLUME_SPEC	31
2.2.11	VOLUME_INFO	31
2.2.12	DISK_SPEC.....	32
2.2.13	REGION_SPEC.....	33
2.2.14	DRIVE_LETTER_INFO	33
2.2.15	FILE_SYSTEM_INFO	34
2.2.16	IFILE_SYSTEM_INFO	36
2.2.17	TASK_INFO.....	38
2.2.18	DMPROGRESS_TYPE.....	39
2.2.19	COUNTED_STRING.....	39
2.2.20	MERGE_OBJECT_INFO.....	40
2.3	IVolumeClient Interface	41
2.3.1	IVolumeClient Data Types	41
2.3.1.1	PARTITION_OS2_BOOT Constant.....	41
2.3.1.2	DISK_INFO.....	41
2.3.1.3	REGION_INFO	44
2.4	IVolumeClient2 Interface.....	46
2.4.1	IVolumeClient2 Data Types	46
2.5	IVolumeClient3 Interface.....	46
2.5.1	IVolumeClient3 Data Types	46
2.5.1.1	PARTITIONSTYLE	46
2.5.1.2	DISK_INFO_EX	46
2.5.1.3	REGION_INFO_EX	51
2.6	IVolumeClient4 Interface.....	53
2.6.1	IVolumeClient4 Data Types	53
2.7	IDMRemoteServer Interface.....	53
2.7.1	IDMRemoteServer Data Types	53

2.8	IDMNotify Interface	53
2.8.1	IDMNotify Data Types.....	54
2.8.1.1	DMNOTIFY_INFO_TYPE	54
2.8.1.2	LDMACTION	54
3	Protocol Details	56
3.1	Client Role Details	56
3.1.1	Abstract Data Model	56
3.1.2	Timers	56
3.1.3	Initialization	56
3.1.4	Message Processing and Sequencing Rules	56
3.1.4.1	Higher-Layer Triggered Events	62
3.1.4.1.1	Common Details	62
3.1.4.1.1.1	Methods with Prerequisites	62
3.1.4.1.1.2	Parameters to IVolumeClient and IVolumeClient3	62
3.1.4.1.1.3	Relationships Between Storage Objects	62
3.1.4.1.2	Drive Letters	63
3.1.4.1.3	File Systems	63
3.1.4.1.4	Disks	64
3.1.4.1.5	Partitions.....	66
3.1.4.1.6	Volumes	67
3.1.4.1.7	Tasks.....	70
3.1.4.1.8	Loss of Connection	70
3.1.4.2	Processing Server Replies to Method Calls.....	70
3.1.4.3	Processing Notifications Sent from the Server to the Client	70
3.1.4.4	Protocol Message Details	71
3.1.4.4.1	IDMNotify Methods.....	71
3.1.4.4.1.1	IDMNotify::ObjectsChanged (Opnum 3).....	71
3.1.5	Timer Events.....	73
3.1.6	Other Local Events	73
3.2	Server Role Details.....	74
3.2.1	Abstract Data Model	74
3.2.1.1	List of Storage Objects Present in the System.....	74
3.2.1.2	List of Clients Connected to the Server	75
3.2.1.3	List of Tasks Currently Executed on the Server	75
3.2.2	Timers	75
3.2.3	Initialization	75
3.2.3.1	List of Storage Objects Present in the System.....	76
3.2.3.2	List of Clients Connected to the Server	76
3.2.3.3	List of Tasks Currently Executed on the Server	76
3.2.4	Message Processing and Sequencing Rules	76
3.2.4.1	Higher-Layer Triggered Events	76
3.2.4.2	Rules for Modifying the List of Storage Objects	76
3.2.4.3	Rules for Handling Synchronous and Asynchronous Tasks.....	77
3.2.4.4	Protocol Message Details	79
3.2.4.4.1	IVolumeClient Methods	79
3.2.4.4.1.1	IVolumeClient::EnumDisks (Opnum 3)	82
3.2.4.4.1.2	IVolumeClient::EnumDiskRegions (Opnum 4)	83
3.2.4.4.1.3	IVolumeClient::CreatePartition (Opnum 5)	83
3.2.4.4.1.4	IVolumeClient::CreatePartitionAssignAndFormat (Opnum 6)	85
3.2.4.4.1.5	IVolumeClient::CreatePartitionAssignAndFormatEx (Opnum 7).....	87
3.2.4.4.1.6	IVolumeClient::DeletePartition (Opnum 8)	88
3.2.4.4.1.7	IVolumeClient::WriteSignature (Opnum 9)	90
3.2.4.4.1.8	IVolumeClient::MarkActivePartition (Opnum 10)	91
3.2.4.4.1.9	IVolumeClient::Eject (Opnum 11)	92

3.2.4.4.1.10	IVolumeClient::FTEnumVolumes (Opnum 13)	93
3.2.4.4.1.11	IVolumeClient::FTEnumLogicalDiskMembers (Opnum 14)	94
3.2.4.4.1.12	IVolumeClient::FTDeleteVolume (Opnum 15)	95
3.2.4.4.1.13	IVolumeClient::FTBreakMirror (Opnum 16)	97
3.2.4.4.1.14	IVolumeClient::FTResyncMirror (Opnum 17)	99
3.2.4.4.1.15	IVolumeClient::FTRegenerateParityStripe (Opnum 18)	100
3.2.4.4.1.16	IVolumeClient::FTReplaceMirrorPartition (Opnum 19)	101
3.2.4.4.1.17	IVolumeClient::FTReplaceParityStripePartition (Opnum 20)	103
3.2.4.4.1.18	IVolumeClient::EnumDriveLetters (Opnum 21)	105
3.2.4.4.1.19	IVolumeClient::AssignDriveLetter (Opnum 22)	106
3.2.4.4.1.20	IVolumeClient::FreeDriveLetter (Opnum 23)	108
3.2.4.4.1.21	IVolumeClient::EnumLocalFileSystems (Opnum 24)	110
3.2.4.4.1.22	IVolumeClient::GetInstalledFileSystems (Opnum 25)	111
3.2.4.4.1.23	IVolumeClient::Format (Opnum 26)	112
3.2.4.4.1.24	IVolumeClient::EnumVolumes (Opnum 28)	113
3.2.4.4.1.25	IVolumeClient::EnumVolumeMembers (Opnum 29)	114
3.2.4.4.1.26	IVolumeClient::CreateVolume (Opnum 30)	115
3.2.4.4.1.27	IVolumeClient::CreateVolumeAssignAndFormat (Opnum 31)	117
3.2.4.4.1.28	IVolumeClient::CreateVolumeAssignAndFormatEx (Opnum 32)	119
3.2.4.4.1.29	IVolumeClient::GetVolumeMountName (Opnum 33)	120
3.2.4.4.1.30	IVolumeClient::GrowVolume (Opnum 34)	121
3.2.4.4.1.31	IVolumeClient::DeleteVolume (Opnum 35)	123
3.2.4.4.1.32	IVolumeClient::AddMirror (Opnum 36)	125
3.2.4.4.1.33	IVolumeClient::RemoveMirror (Opnum 37)	127
3.2.4.4.1.34	IVolumeClient::SplitMirror (Opnum 38)	128
3.2.4.4.1.35	IVolumeClient::InitializeDisk (Opnum 39)	130
3.2.4.4.1.36	IVolumeClient::UninitializeDisk (Opnum 40)	131
3.2.4.4.1.37	IVolumeClient::ReConnectDisk (Opnum 41)	133
3.2.4.4.1.38	IVolumeClient::ImportDiskGroup (Opnum 43)	134
3.2.4.4.1.39	IVolumeClient::DiskMergeQuery (Opnum 44)	136
3.2.4.4.1.40	IVolumeClient::DiskMerge (Opnum 45)	138
3.2.4.4.1.41	IVolumeClient::ReAttachDisk (Opnum 47)	139
3.2.4.4.1.42	IVolumeClient::ReplaceDisk (Opnum 48)	141
3.2.4.4.1.43	IVolumeClient::ReplaceRaid5Column (Opnum 51)	142
3.2.4.4.1.44	IVolumeClient::RestartVolume (Opnum 52)	144
3.2.4.4.1.45	IVolumeClient::GetEncapsulateDiskInfo (Opnum 53)	145
3.2.4.4.1.46	IVolumeClient::EncapsulateDisk (Opnum 54)	149
3.2.4.4.1.47	IVolumeClient::QueryChangePartitionNumbers (Opnum 55)	152
3.2.4.4.1.48	IVolumeClient::DeletePartitionNumberInfoFromRegistry (Opnum 56)	153
3.2.4.4.1.49	IVolumeClient::SetDontShow (Opnum 57)	153
3.2.4.4.1.50	IVolumeClient::GetDontShow (Opnum 58)	154
3.2.4.4.1.51	IVolumeClient::EnumTasks (Opnum 67)	154
3.2.4.4.1.52	IVolumeClient::GetTaskDetail (Opnum 68)	155
3.2.4.4.1.53	IVolumeClient::AbortTask (Opnum 69)	156
3.2.4.4.1.54	IVolumeClient::HrGetErrorData (Opnum 70)	157
3.2.4.4.1.55	IVolumeClient::Initialize (Opnum 71)	158
3.2.4.4.1.56	IVolumeClient::Uninitialize (Opnum 72)	160
3.2.4.4.1.57	IVolumeClient::Refresh (Opnum 73)	160
3.2.4.4.1.58	IVolumeClient::RescanDisks (Opnum 74)	161
3.2.4.4.1.59	IVolumeClient::RefreshFileSys (Opnum 75)	161
3.2.4.4.1.60	IVolumeClient::SecureSystemPartition (Opnum 76)	162
3.2.4.4.1.61	IVolumeClient::ShutDownSystem (Opnum 77)	162
3.2.4.4.1.62	IVolumeClient::EnumAccessPath (Opnum 78)	162
3.2.4.4.1.63	IVolumeClient::EnumAccessPathForVolume (Opnum 79)	163
3.2.4.4.1.64	IVolumeClient::AddAccessPath (Opnum 80)	164

3.2.4.4.1.65	IVolumeClient::DeleteAccessPath (Opnum 81).....	165
3.2.4.4.2	IVolumeClient2.....	166
3.2.4.4.2.1	IVolumeClient2::GetMaxAdjustedFreeSpace (Opnum 3)	166
3.2.4.4.3	IVolumeClient3.....	167
3.2.4.4.3.1	IVolumeClient3::EnumDisksEx (Opnum 3)	170
3.2.4.4.3.2	IVolumeClient3::EnumDiskRegionsEx (Opnum 4)	171
3.2.4.4.3.3	IVolumeClient3::CreatePartition (Opnum 5)	171
3.2.4.4.3.4	IVolumeClient3::CreatePartitionAssignAndFormat (Opnum 6)	172
3.2.4.4.3.5	IVolumeClient3::CreatePartitionAssignAndFormatEx (Opnum 7)	173
3.2.4.4.3.6	IVolumeClient3::DeletePartition (Opnum 8)	174
3.2.4.4.3.7	IVolumeClient3::InitializeDiskStyle (Opnum 9)	174
3.2.4.4.3.8	IVolumeClient3::MarkActivePartition (Opnum 10)	176
3.2.4.4.3.9	IVolumeClient3::Eject (Opnum 11)	176
3.2.4.4.3.10	IVolumeClient3::FTEnumVolumes (Opnum 13)	177
3.2.4.4.3.11	IVolumeClient3::FTEnumLogicalDiskMembers (Opnum 14)	177
3.2.4.4.3.12	IVolumeClient3::FTDeleteVolume (Opnum 15).....	178
3.2.4.4.3.13	IVolumeClient3::FTBreakMirror (Opnum 16).....	178
3.2.4.4.3.14	IVolumeClient3::FTResyncMirror (Opnum 17).....	179
3.2.4.4.3.15	IVolumeClient3::FTRegenerateParityStripe (Opnum 18).....	179
3.2.4.4.3.16	IVolumeClient3::FTReplaceMirrorPartition (Opnum 19)	180
3.2.4.4.3.17	IVolumeClient3::FTReplaceParityStripePartition (Opnum 20).....	181
3.2.4.4.3.18	IVolumeClient3::EnumDriveLetters (Opnum 21)	182
3.2.4.4.3.19	IVolumeClient3::AssignDriveLetter (Opnum 22).....	182
3.2.4.4.3.20	IVolumeClient3::FreeDriveLetter (Opnum 23).....	183
3.2.4.4.3.21	IVolumeClient3::EnumLocalFileSystems (Opnum 24).....	184
3.2.4.4.3.22	IVolumeClient3::GetInstalledFileSystems (Opnum 25)	184
3.2.4.4.3.23	IVolumeClient3::Format (Opnum 26)	185
3.2.4.4.3.24	IVolumeClient3::EnumVolumes (Opnum 27)	186
3.2.4.4.3.25	IVolumeClient3::EnumVolumeMembers (Opnum 28)	186
3.2.4.4.3.26	IVolumeClient3::CreateVolume (Opnum 29).....	186
3.2.4.4.3.27	IVolumeClient3::CreateVolumeAssignAndFormat (Opnum 30).....	187
3.2.4.4.3.28	IVolumeClient3::CreateVolumeAssignAndFormatEx (Opnum 31)	188
3.2.4.4.3.29	IVolumeClient3::GetVolumeMountName (Opnum 32)	189
3.2.4.4.3.30	IVolumeClient3::GrowVolume (Opnum 33).....	190
3.2.4.4.3.31	IVolumeClient3::DeleteVolume (Opnum 34)	190
3.2.4.4.3.32	IVolumeClient3::CreatePartitionsForVolume (Opnum 35)	191
3.2.4.4.3.33	IVolumeClient3::DeletePartitionsForVolume (Opnum 36).....	193
3.2.4.4.3.34	IVolumeClient3::GetMaxAdjustedFreeSpace (Opnum 37)	194
3.2.4.4.3.35	IVolumeClient3::AddMirror (Opnum 38).....	194
3.2.4.4.3.36	IVolumeClient3::RemoveMirror (Opnum 39).....	195
3.2.4.4.3.37	IVolumeClient3::SplitMirror (Opnum 40).....	196
3.2.4.4.3.38	IVolumeClient3::InitializeDiskEx (Opnum 41)	196
3.2.4.4.3.39	IVolumeClient3::UninitializeDisk (Opnum 42)	198
3.2.4.4.3.40	IVolumeClient3::ReConnectDisk (Opnum 43)	198
3.2.4.4.3.41	IVolumeClient3::ImportDiskGroup (Opnum 44)	199
3.2.4.4.3.42	IVolumeClient3::DiskMergeQuery (Opnum 45)	199
3.2.4.4.3.43	IVolumeClient3::DiskMerge (Opnum 46).....	200
3.2.4.4.3.44	IVolumeClient3::ReAttachDisk (Opnum 47).....	201
3.2.4.4.3.45	IVolumeClient3::ReplaceRaid5Column (Opnum 48)	201
3.2.4.4.3.46	IVolumeClient3::RestartVolume (Opnum 49).....	202
3.2.4.4.3.47	IVolumeClient3::GetEncapsulateDiskInfoEx (Opnum 50)	202
3.2.4.4.3.48	IVolumeClient3::EncapsulateDiskEx (Opnum 51)	206
3.2.4.4.3.49	IVolumeClient3::QueryChangePartitionNumbers (Opnum 52).....	209
3.2.4.4.3.50	IVolumeClient3::DeletePartitionNumberInfoFromRegistry (Opnum 53)	210
3.2.4.4.3.51	IVolumeClient3::SetDontShow (Opnum 54)	210

3.2.4.4.3.52	IVolumeClient3::GetDontShow (Opnum 55)	210
3.2.4.4.3.53	IVolumeClient3::EnumTasks (Opnum 64)	211
3.2.4.4.3.54	IVolumeClient3::GetTaskDetail (Opnum 65)	211
3.2.4.4.3.55	IVolumeClient3::AbortTask (Opnum 66)	212
3.2.4.4.3.56	IVolumeClient3::HrGetErrorData (Opnum 67)	212
3.2.4.4.3.57	IVolumeClient3::Initialize (Opnum 68)	213
3.2.4.4.3.58	IVolumeClient3::Uninitialize (Opnum 69)	214
3.2.4.4.3.59	IVolumeClient3::Refresh (Opnum 70)	214
3.2.4.4.3.60	IVolumeClient3::RescanDisks (Opnum 71)	214
3.2.4.4.3.61	IVolumeClient3::RefreshFileSys (Opnum 72)	215
3.2.4.4.3.62	IVolumeClient3::SecureSystemPartition (Opnum 73)	215
3.2.4.4.3.63	IVolumeClient3::ShutDownSystem (Opnum 74)	215
3.2.4.4.3.64	IVolumeClient3::EnumAccessPath (Opnum 75)	216
3.2.4.4.3.65	IVolumeClient3::EnumAccessPathForVolume (Opnum 76)	216
3.2.4.4.3.66	IVolumeClient3::AddAccessPath (Opnum 77)	217
3.2.4.4.3.67	IVolumeClient3::DeleteAccessPath (Opnum 78)	217
3.2.4.4.4	IVolumeClient4	217
3.2.4.4.4.1	IVolumeClient4::RefreshEx (Opnum 3)	218
3.2.4.4.4.2	IVolumeClient4::GetVolumeDeviceName (Opnum 4)	218
3.2.4.4.5	IDMRemoteServer	219
3.2.4.4.5.1	IDMRemoteServer::CreateRemoteObject (Opnum 3)	219
3.2.5	Timer Events	220
3.2.6	Other Local Events	220
3.2.6.1	Disk Arrival	220
3.2.6.2	Disk Removal	220
3.2.6.3	Disk Layout Change	221
3.2.6.4	File System Change	221
3.2.6.5	Drive Letter Arrival	221
3.2.6.6	Drive Letter Removal	221
3.2.6.7	Media Arrival	221
3.2.6.8	Media Removal	221
4	Protocol Examples	223
4.1	Starting a New Session on a Local or Remote Server	223
4.2	Starting a New Session on a Remote Server Using the IDMRemoteServer Interface	224
4.3	Creating a Partition	226
4.4	Deleting a Partition	228
4.5	Creating a Volume	230
4.6	Deleting a Volume	232
5	Security Considerations	235
6	Appendix A: Full IDL	236
6.1	dmintf.idl	236
6.2	dmintf3.idl	236
6.3	unknown.idl	244
7	Appendix B: Windows Behavior	246
8	Appendix C: IDMNotify::ObjectsChanged	255
9	Index	262

1 Introduction

The Disk Management Remote Protocol is a set of Distributed Component Object Model (DCOM) interfaces, as specified in [\[MS-DCOM\]](#), built for managing storage objects on a machine. The Disk Management Remote Protocol relies on detailed, low-level operating system and storage concepts. While the basic concepts are outlined in this specification, it assumes that the reader has familiarity with these technologies.

For background information on storage, **disk**, and **volume** concepts, see [\[MSDN-DISKMAN\]](#) and [\[MSDN-VOLMAN\]](#). For the IDL specification, see sections [6.1](#) and [6.2](#).

1.1 Glossary

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

- Active Partition**
- Allocation Unit Size**
- ASCII**
- Basic Disk**
- Boot File**
- Boot Loader**
- Boot Loader File**
- Boot Partition**
- Boot Volume**
- Boot.ini**
- Bus**
- Bus Type**
- Compact Disc File System (CDFS)**
- Crash Dump File**
- Cylinder**
- Disk**
- Disk Adapter**
- Disk Extent**
- Disk Group**
- Disk Group Import**
- Disk Regions**
- Disk Signature**
- Disk Type**
- Drive Letter**
- Dynamic Volume**
- Encapsulate**
- Extended Partition**
- Fault-Tolerant**
- Fault-Tolerant Mirror Set**
- File**
- File Allocation Units**
- File System**
- File System Flags**
- Flags**
- Folder**
- Foreign**
- Format**
- Free Space**
- Full Format**
- GUID Partition Table (GPT)**

Hard Disk Physical Name
Hibernation Image
Integrated Drive Electronics (IDE) Bus
Locked Partition
Logical Disk Manager (LDM)
Logical Drive
Logical Partition
Mass Storage Device
Master Boot Record (MBR)
Microsoft Interface Definition Language (MIDL)
Mirrored Volume
Modification Sequence Number
Mount Path
Mount Point
Mounted Folder
New Technology File System (NTFS)
Offline
Online
Paging File
Partition
Partition Table (also GPT)
Partition Type
Path
Primary Disk Group
Primary Partition
Quick Formatting
RAID
RAID-5
Region
Region Flags
Region's Status
Remote Procedure Call (RPC)
Removable Media
RPC Protocol Sequence
SCSI Logical Unit Number (LUN)
SCSI Port Number
Sector
Serial Storage Architecture (SSA) Bus
Signature
Simple Volume
Small Computer System Interface (SCSI) Bus
System Directory
System Partition
Track
Region
Unicode
Universal Disk Format (UDF)
Universal Serial Bus (USB)
Universally Unique Identifier (UUID) or Globally Unique Identifier (GUID)
Volume
Volume Data
Volume Label
Volume Manager
Volume Members

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

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

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

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

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

[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-VDS] Microsoft Corporation, "[Virtual Disk Service \(VDS\) Protocol Specification](#)", March 2007.

[MSDN-DISKMAN] Microsoft Corporation, "Disk Management", <http://msdn2.microsoft.com/en-us/library/aa363978.aspx>

[MSDN-STC] Microsoft Corporation, "Storage Technologies Collection", March 2003, <http://technet2.microsoft.com/WindowsServer/en/Library/616e5e77-958b-42f0-a87f-ba229ccd81721033.msp>

[MSDN-VOLMAN] Microsoft Corporation, "Volume Management", <http://msdn2.microsoft.com/en-us/library/aa365728.aspx>

[MSDN-PARTITIONINFO] Microsoft Corporation, "PARTITION_INFORMATION_EX", <http://msdn2.microsoft.com/en-us/library/aa365448.aspx>

1.3 Protocol Overview (Synopsis)

The Disk Management Remote Protocol provides a set of DCOM interfaces for managing storage objects, such as disks and volumes. The protocol also enables clients to obtain notifications of changes to storage objects. The server end of the protocol is a storage management service that

implements supports for the DCOM interface. The client end of the protocol is an application that invokes method calls on the interface to perform various disk and volume configuration tasks.

This protocol includes the following six DCOM interfaces:

- [IVolumeClient](#)
- [IVolumeClient2](#)
- [IVolumeClient3](#)
- [IVolumeClient4](#)
- [IDMRemoteServer](#)
- [IDMNotify](#)

The IVolumeClient and IVolumeClient2 interfaces provide methods for managing storage objects, such as disks and volumes.

IVolumeClient3 supersedes IVolumeClient and IVolumeClient2, and contains new functionality related to the **GUID partition table (GPT)** disk partitioning style.

IVolumeClient4 includes additional functionality to augment what is provided by IVolumeClient3; it verifies that disk access and disk media record information is valid when refreshing the cache, and it queries the device **path** for a volume.

IDMRemoteServer includes functionality to create an instance of disk management server on a remote machine.

IDMNotify is the interface implemented by the client to receive notifications from the disk management server. [<1>](#)

1.4 Relationship to Other Protocols

The Disk Management Remote Protocol relies on the [DCOM Remote Protocol](#) (as specified in [MS-DCOM]), which uses **RPC** (as specified in [\[MS-RPCE\]](#)) as its transport. The Disk Management Remote Protocol is not used by any other protocols. [<2>](#)

1.5 Prerequisites/Preconditions

This protocol is implemented over DCOM and RPC, and, as a result, has the prerequisites specified in [\[MS-DCOM\]](#) and [\[MS-RPCE\]](#) as being common to DCOM and RPC interfaces.

The Disk Management Remote Protocol assumes that a client has obtained the name of a server that supports this protocol suite before the protocol is invoked. The protocol also assumes that the client has sufficient security privileges to configure disks and volumes on the server.

1.6 Applicability Statement

This protocol is applicable when an application needs to remotely configure disks and volumes.

The [Virtual Disk Service \(VDS\) Remote Protocol](#) may also be used to perform logical functions similar to those performed by this protocol. For more information, see [MS-VDS]. [<3>](#)

1.7 Versioning and Capability Negotiation

Supported Transports: This protocol uses the [DCOM Remote Protocol](#) (as specified in [MS-DCOM]), which in turn uses RPC over TCP, as its only transport. For more information, see section [2.1](#).

Protocol Version: This protocol includes six DCOM interfaces, all of which MUST be version 0.0.

The client negotiates for a given set of server functionality by specifying the **UUID** that corresponds to the wanted RPC interface when binding to the server. [<4>](#)

Security and Authentication Methods: As specified in [MS-DCOM], [\[MS-RPCE\]](#), and section [2.1](#).

1.8 Vendor-Extensible Fields

This protocol does not define any vendor-extensible fields.

1.9 Standards Assignments

Parameter	Value	Reference
RPC Interface UUID for IVolumeClient	D2D79DF5-3400-11d0-B40B-00AA005FF586	Section 2.1
RPC Interface UUID for IVolumeClient2	4BDAFC52-FE6A-11d2-93F8-00105A11164A	Section 2.1
RPC Interface UUID for IVolumeClient3	135698D2-3A37-4d26-99DF-E2BB6AE3AC61	Section 2.1
RPC Interface UUID for IVolumeClient4	DEB01010-3A37-4d26-99DF-E2BB6AE3AC61	Section 2.1
RPC Interface UUID for IDMRRemoteServer	3A410F21-553F-11d1-8E5E-00A0C92C9D5D	Section 2.1
RPC Interface UUID for IDMNotify	D2D79DF7-3400-11d0-B40B-00AA005FF586	Section 2.1

2 Messages

The following sections specify how Disk Management Remote Protocol messages are transported and common data types.

2.1 Transport

This protocol MUST use the [DCOM Remote Protocol](#), as specified in [MS-DCOM], as its transport. On its behalf, the DCOM Remote Protocol uses the following **RPC protocol sequence**: RPC over TCP, as specified in [\[MS-RPCE\]](#).

An implementation of Disk Management MAY configure its DCOM implementation or underlying RPC transport with authentication parameters to allow clients to connect. The details of this are implementation-specific. [<5>](#)

The Disk Management interfaces make use of the underlying DCOM security framework, as specified in [MS-DCOM], and rely upon it for access control. DCOM distinguishes between launch and access. An implementation of Disk Management SHOULD differentiate between these two types and impose different authorization requirements per interface. The IVolumeClient, IVolumeClient2, IVolumeClient3, and IVolumeClient4 interfaces SHOULD be the most restrictive, requiring the invoker to have access to the Disk Management system. The IDMRemoteServer interface SHOULD be less restrictive, because it provides less access to the underlying system. [<6>](#)

2.2 Common Data Types

In addition to RPC base types and definitions (as specified in [\[C706\]](#) and [\[MS-RPCE Appendix A\]](#)), the following sections use the definitions of [LONGLONG](#), [DWORD](#), [Boolean](#), [BYTE](#), [LONG](#), [WCHAR](#), [LPSTR](#), [BOOL](#), [FILETIME](#), [GUID](#), and [ULONG](#), which are specified in [\[MS-DTYP\]](#).

This section defines a number of fields that contain **flags** that are combined by using a logical OR operation. Except where otherwise specified, all undefined flags MUST be set to 0 and ignored on receipt.

For all methods that have an array as an output parameter, memory for the array is allocated by the server and freed by the client. Details about DCOM memory allocation mechanisms are as specified in [\[MS-DCOM\]](#).

2.2.1 HRESULT Return Codes

The following **HRESULT** return codes are defined by the Disk Management Remote Protocol, and together with the [HRESULTs](#), as specified in [\[MS-ERREF\]](#), they MAY be returned by the server to indicate additional information about the result of a method call or the reason a call failed. If the result is an error rather than simple status information, the most significant bit of the **HRESULT** is set (as specified in [MS-ERREF]).

Return value/code	Description
0xC1000001L LDM_E_UNEXPECTED	An unexpected error has occurred. Check the system event log for more information on the error. Close the Disk Management console, then restart Disk Management or restart the computer.
0xC1000004L	INTERNAL Error - The commit status has been

Return value/code	Description
LDM_E_LOST	lost.
0xC100000AL LDM_E_BADMSG	INTERNAL Error - The request contains a message format that is not valid.
0xC100000CL LDM_E_NOMEM	There is not enough memory available to complete the operation. Save your work, exit other programs, and then try again.
0xC1000014L LDM_E_NOTRANS	INTERNAL Error - The operation requires a transaction.
0xC1000016L LDM_E_RESTART	INTERNAL Error - The commit has been aborted, transaction must be restarted.
0xC1000017L LDM_E_DB_RESTART	The dynamic disks present in the system cannot be configured due to excessive disk I/O errors. Verify the status of storage devices, and then restart the computer.
0xC1000018L LDM_E_VOLDERR	INTERNAL Error - An unexpected error occurred in the configuration program.
0xC1000019L LDM_E_REPLAY	INTERNAL Error - An unexpected error occurred in the kernel during a configuration update.
0xC100001BL LDM_E_LOGIO_FAIL	INTERNAL Error - Logical Disk Manager cannot start the logging thread.
0xC100001CL LDM_E_DG_DISABLED	The disk group has been disabled.
0xC1000029L LDM_E_DIAG_DBASE	One or more errors exist in the disk group configuration copies.
0xC100002DL LDM_E_DIAG_DISABLED	INTERNAL Error - The configuration program is currently disabled.
0xC1000030L LDM_E_DIAG_CYCLE	INTERNAL Error - No convergence between disk group and disk list.
0xC1000032L LDM_E_VB_NOENT	The expected Logical Disk Manager registry keys were not found.
0xC1000033L LDM_E_VB_SYS	A system error was encountered while reading or setting a Logical Disk Manager registry key.
0xC1000034L LDM_E_VB_FORMAT	INTERNAL Error - String format errors were found in Logical Disk Manager registry keys.
0xC100003CL LDM_E_INPROGRESS	INTERNAL Error - A transaction is already in progress.
0xC100003DL LDM_E_DISABLED	INTERNAL Error - This feature is disabled or is not implemented.
0xC100003EL	INTERNAL Error - The requested operation is

Return value/code	Description
LDM_E_NOSUPPORT	not supported.
0xC100003FL LDM_E_LOCK	INTERNAL Error - The requested lock could not be obtained.
0xC1000040L LDM_E_NOLOCK	INTERNAL Error - The required lock is not held in the transaction.
0xC1000041L LDM_E_NO_DLOCK	INTERNAL Error - The required data lock is not held in the transaction.
0xC1000042L LDM_E_EXIST	INTERNAL Error - An object with the specified name already exists in the disk group.
0xC1000043L LDM_E_NOENT	The specified object no longer exists in the disk group.
0xC1000044L LDM_E_CONFIG	INTERNAL Error - The disk group configuration has changed.
0xC1000046L LDM_E_BUSY	The specified object is in active use, operation not allowed.
0xC1000047L LDM_E_INVALID	The requested operation is not valid.
0xC1000049L LDM_E_NEG_SIZE	INTERNAL Error - Negative length, width, or offset received as parameter of the request.
0xC100004BL LDM_E_BADNAME	INTERNAL Error - The record name is not valid.
0xC100004CL LDM_E_NOT_DIS	The operation requires a disabled volume.
0xC100004DL LDM_E_ENABLED	This mirror cannot be removed because the mirrored volume is currently missing or regenerating. Retry this operation when the mirror state is healthy.
0xC100004EL LDM_E_NO_ASSOC	INTERNAL Error - The operation requires an associated record.
0xC100004FL LDM_E_ASSOC	The operation is not allowed because the requested object is in active use.
0xC1000052L LDM_E_HOLE	INTERNAL Error - The specified striped plex is not compact.
0xC1000053L LDM_E_DIFF_SIZE	INTERNAL Error - The subdisks of the striped plex have different sizes.
0xC1000054L LDM_E_NONE_ASSOC	INTERNAL Error - The striped plex has no associated subdisks.
0xC1000055L LDM_E_FULL_ASSOC	INTERNAL Error - The record cannot have more associations.

Return value/code	Description
0xC1000056L LDM_E_INVALID_RTYPE	INTERNAL Error - The record type is not valid.
0xC1000057L LDM_E_NOT_SUBDISK	INTERNAL Error - The specified record is not a subdisk.
0xC1000058L LDM_E_NOT_PLEX	INTERNAL Error - The specified record is not a plex.
0xC1000059L LDM_E_NOT_VOL	INTERNAL Error - The specified record is not a volume.
0xC100005AL LDM_E_NOT_DM	INTERNAL Error - The specified record is not a disk media.
0xC100005BL LDM_E_NOT_DG	INTERNAL Error - The specified record is not a disk group.
0xC100005CL LDM_E_NOT_DA	INTERNAL Error - The specified record is not a disk access.
0xC100005DL LDM_E_IS_DG	INTERNAL Error - The operation is not allowed on a disk group.
0xC100005EL LDM_E_MAX_VOL	The volume limit has been reached, no additional dynamic volumes can be created.
0xC100005FL LDM_E_MAX_PLEX	INTERNAL Error - Too many plexes exist.
0xC1000060L LDM_E_OVERLAP	INTERNAL Error - The subdisk overlaps with another subdisk.
0xC1000061L LDM_E_TOO_SMALL	INTERNAL Error - A striped plex subdisk has the length shorter than the stripe width.
0xC1000062L LDM_E_SD_WIDTH	INTERNAL Error - A striped plex subdisk has the length not multiple of the stripe width.
0xC1000063L LDM_E_INVALID_FIELD	INTERNAL Error - The specified field is not valid.
0xC1000064L LDM_E_ST_WIDTH	INTERNAL Error - The plex stripe width is not valid.
0xC1000065L LDM_E_OVERFLOW	INTERNAL Error - The operation overflows the maximum offsets.
0xC1000066L LDM_E_LOG_SD_SMALL	INTERNAL Error - The log subdisk is too small for the volume.
0xC1000069L LDM_E_HAS_LOG	INTERNAL Error - The plex already has a log subdisk.
0xC100006BL LDM_E_NO_DG	The specified disk group no longer exists.
0xC100006CL	INTERNAL Error - The request crosses a disk

Return value/code	Description
LDM_E_LDM_E_DG_MISMATCH	group boundary.
0xC100006DL LDM_E_NOT_ROOTDG	INTERNAL Error - The request is allowed only in the root disk group.
0xC100006EL LDM_E_PUB_BOUNDS	INTERNAL Error - The subdisk is not within the public region boundaries.
0xC100006FL LDM_E_BAD_DISK	The specified disk is not ready or usable.
0xC1000070L LDM_E_VOL_INVALID	INTERNAL Error - The specified volume is not usable.
0xC1000071L LDM_E_PATH_STAT	The specified device path cannot be accessed.
0xC1000073L LDM_E_NOROOTDG	INTERNAL Error - The root disk group is not enabled.
0xC1000074L LDM_E_NODATATYPE	INTERNAL Error - The disk access type is not recognized.
0xC1000075L LDM_E_DAOFFLINE	The specified disk is offline .
0xC1000076L LDM_E_REC_TOOBIG	INTERNAL Error - The record store size is too large.
0xC1000077L LDM_E_SD_BAD	INTERNAL Error - The specified subdisk is not enabled.
0xC1000078L LDM_E_PLEX_BAD	INTERNAL Error - The specified plex contains disabled subdisks.
0xC1000079L LDM_E_NOT_EMPTY	The operation is not allowed because the specified disk is not empty.
0xC100007AL LDM_E_DG_EXIST	INTERNAL Error - A disk group with the specified name already exists.
0xC100007BL LDM_E_DGCREATED	INTERNAL Error - The operation is not allowed on a created disk group.
0xC100007CL LDM_E_DG_LOG_FULL	INTERNAL Error - The configuration is too large for the disk group log.
0xC100007DL LDM_E_DG_LOG_SMALL	INTERNAL Error - The disk log is too small for the disk group configuration.
0xC100007EL LDM_E_DG_IMPORTED	INTERNAL Error - The disk group has already been imported.
0xC100007FL LDM_E_DB_REC_ERROR	INTERNAL Error - An error exists in the configuration record.
0xC1000080L LDM_E_DB_TOO_BIG	INTERNAL Error - The configuration is too large to make configuration copies.

Return value/code	Description
0xC1000081L LDM_E_DGNOTCREATED	INTERNAL Error - The disk group creation is not complete.
0xC1000082L LDM_E_LOG_BAD	INTERNAL Error - There are no valid log copies in the disk group.
0xC1000083L LDM_E_ROOTDG	INTERNAL Error - The operation is not allowed on the root disk group.
0xC1000084L LDM_E_LAST_DISK	INTERNAL Error - Logical Disk Manager cannot remove the last disk in the disk group.
0xC1000085L LDM_E_LAST_CONFIG	INTERNAL Error - Logical Disk Manager cannot remove the last disk group configuration copy.
0xC1000086L LDM_E_LAST_LOG	INTERNAL Error - Logical Disk Manager cannot remove the last disk group log copy.
0xC1000087L LDM_E_DB_FULL	The Logical Disk Manager configuration is full. No more objects can be created.
0xC1000088L LDM_E_DB_NOENT	INTERNAL Error - The database file cannot be found.
0xC1000089L LDM_E_DB_SYS	INTERNAL Error - System error in configuration copy.
0xC100008AL LDM_E_DB_FORMAT	INTERNAL Error - A format error was found in the configuration copy.
0xC100008BL LDM_E_DB_VERSION	The configuration format version is not supported.
0xC100008CL LDM_E_DB_NOSPC	INTERNAL Error - The Logical Disk Manager configuration is full.
0xC100008DL LDM_E_DB_SHORT	INTERNAL Error - Unexpected end of the configuration copy.
0xC100008EL LDM_E_DB_READ	Logical Disk Manager could not read the disk configuration.
0xC100008FL LDM_E_DB_WRITE	Logical Disk Manager could not write the disk configuration.
0xC1000090L LDM_E_DB_AS_BAD	INTERNAL Error - Association not resolved
0xC1000091L LDM_E_DB_ASCNT_BAD	INTERNAL Error - The association count is incorrect.
0xC1000094L LDM_E_DB_MAGIC	INTERNAL Error - Invalid magic number in the configuration copy.
0xC1000095L LDM_E_DB_BLKNO	INTERNAL Error - Invalid block number in the configuration copy.

Return value/code	Description
0xC1000096L LDM_E_DB_NOMATCH	INTERNAL Error - No valid disk belonging to the disk group was found.
0xC1000097L LDM_E_DB_DUPLICATE	INTERNAL Error - Duplicate records exist in the configuration.
0xC1000098L LDM_E_DB_INCONSISTENT	INTERNAL Error - The configuration records are inconsistent.
0xC1000099L LDM_E_DB_NO_DGREC	INTERNAL Error - No disk group record exists in the configuration.
0xC100009AL LDM_E_DB_BAD_TEMP	INTERNAL Error - The temporary and permanent configurations do not match.
0xC100009BL LDM_E_DB_CHANGED	INTERNAL Error - The on-disk configuration changed during recovery.
0xC100009EL LDM_E_KERN_KNOENT	INTERNAL Error - A record that was expected was not found in the kernel.
0xC100009FL LDM_E_KERN_DBNOENT	INTERNAL Error - A record that exists in the kernel was not found in the configuration.
0xC10000A0L LDM_E_KERN_KDIFF	INTERNAL Error - The configuration record does not match the kernel.
0xC10000A1L LDM_E_KERN_INCONSISTENT	INTERNAL Error - The kernel and on-disk configurations do not match.
0xC10000A2L LDM_E_PUB_SIZE	INTERNAL Error - The public region of the disk is too small.
0xC10000A3L LDM_E_PRIV_SIZE	INTERNAL Error - The private region of the disk is too small.
0xC10000A4L LDM_E_PRIV_FULL	INTERNAL Error - The private region of the disk is full.
0xC10000A5L LDM_E_PRIV_FORMAT	INTERNAL Error - A format error was found in the private region of the disk.
0xC10000A6L LDM_E_PRIV_HEADMATCH	INTERNAL Error - The disk has inconsistent disk headers.
0xC10000A7L LDM_E_PRIV_NOHEADER	INTERNAL Error - The disk header cannot be found.
0xC10000A8L LDM_E_PRIV_INVAL	INTERNAL Error - The disk private region contents are not valid.
0xC10000A9L LDM_E_PRIV_VERSION	INTERNAL Error - The disk private region version is not supported.
0xC10000AAL LDM_E_PRIV_INCONSISTENT	INTERNAL Error - The disks in the disk group are inconsistent.
0xC10000ABL	INTERNAL Error - The attribute cannot be

Return value/code	Description
LDM_E_FIELD_NOREINIT	changed by reinitializing the disk.
0xC10000ACL LDM_E_DISK_INITED	INTERNAL Error - The disk has already been initialized.
0xC10000ADL LDM_E_DISK_ALIASED	INTERNAL Error - The disk header indicates aliased partitions .
0xC10000AEL LDM_E_DISK_SHARED	The disk is marked as shared. It cannot be used in the base Logical Disk Management product.
0xC10000AFL LDM_E_DISK_WRONGHOST	The disk is marked as in use by another computer.
0xC10000B1L LDM_E_MISSING_PUB	INTERNAL Error - The disk has no public partition.
0xC10000B5L LDM_E_BAD_SCTR_SIZE	The disk sector size is not supported by Logical Disk Manager.
0xC10000B6L LDM_E_NO_CONFIGS	INTERNAL Error - The disk group contains no valid configuration copies.
0xC10000B7L LDM_E_DISK_NOT_FOUND	The specified disk cannot be located.
0xC10000B8L LDM_E_DISK_OTHER_DG	INTERNAL Error - The disk belongs to another disk group.
0xC10000B9L LDM_E_BADCOLUMN	INTERNAL Error - The stripe column number is too large for the plex.
0xC10000BAL LDM_E_NOTRAIDVOL	INTERNAL Error - The volume does not have RAID read policy.
0xC10000BBL LDM_E_RAIDVOL	INTERNAL Error - The volume has RAID read policy.
0xC10000BCL LDM_E_TOOMANYRAID	INTERNAL Error - The volume already has one RAID plex.
0xC10000BEL LDM_E_LIC	The license has expired or is not available for the operation.
0xC10000C0L LDM_E_NOTSTORAGE	INTERNAL Error - The volume does not have the storage attribute.
0xC10000C1L LDM_E_SUBVOLUME	INTERNAL Error - The subdisk is defined on a volume.
0xC10000C3L LDM_E_VOLTOOSMALL	INTERNAL Error - The volume length is too small to hold subdisks.
0xC10000C4L LDM_E_HASSUBVOLUME	INTERNAL Error - One or more subdisks are defined on the volume.
0xC10000C5L	The operation is not allowed because it would

Return value/code	Description
LDM_E_BADRAIDCHANGE	make the RAID-5 volume unusable.
0xC10000C6L LDM_E_BADRAID	The operation is not allowed because the RAID-5 volume is currently unusable.
0xC10000C7L LDM_E_PLEX_DIS	INTERNAL Error - The plex is disabled.
0xC10000CCCL LDM_E_MPTH_DISABLE	INTERNAL Error - The record is subsumed by a multipath disk.
0xC10000D1L LDM_E_NT_IF_ERROR	The operation was unsuccessful because a system error occurred.
0xC10000D2L LDM_E_TASK_ABORTED	The operation was canceled at the user's request.
0xC10000D3L LDM_E_TASK_IOERROR	The operation was canceled due to a disk I/O error.
0xC10000D4L LDM_E_TASK_DELETED	INTERNAL Error - The task has been deleted.
0xC10000D5L LDM_E_V_NOT_ENABLED	This operation requires an usable volume.
0xC10000D6L LDM_E_SPC_SETUP_FAIL	There is not enough free disk space to satisfy the request.
0xC10000D7L LDM_E_VOL_SETUP_FAIL	INTERNAL Error - The volume creation setup failed.
0xC10000D8L LDM_E_MIR_SETUP_FAIL	INTERNAL Error - The mirrored volume creation setup failed.
0xC10000DAL LDM_E_GRW_SETUP_FAIL	INTERNAL Error - The volume grow operation setup failed.
0x410000DBL LDM_E_FOREIGN_DA	The operation is not allowed because the disk contains foreign partitions.
0x410000DCL LDM_E_PRIMARY_EXISTS	INTERNAL Error - A primary disk group already exists.
0x410000DDL LDM_E_CAPS_VIOLATION	The requested operation is not allowed in Windows Professional.
0x410000DEL LDM_E_VOL_NO_ZEROING	INTERNAL Error - The operation is not allowed because the volume is still initializing.
0x410000DFL LDM_E_DISK_IN_USE	The requested disk is already in use by this volume.
0x410000E0L LDM_E_SDNODETACH	INTERNAL Error - A subdisk of the column is not detached.
0x410000E1L LDM_E_NOBADDISK	The operation is not needed because no mirrors or RAID-5 members are currently

Return value/code	Description
	disabled.
0x410000E2L LDM_E_DEGRADED	The operation is not needed because no RAID-5 members are currently disabled.
0x410000E3L LDM_E_NO_OPEN	Logical Disk Manager could not open a volume.
0x410000E5L LDM_E_NO_DISMOUNT	Logical Disk Manager could not dismount a volume.
0x410000E6L LDM_E_BAD_PARTITION	Logical Disk Manager could not determine a partition type .
0x410000E7L LDM_E_MNT_FAIL	Logical Disk Manager could not associate a drive letter .
0x410000E8L LDM_E_FT_QUERY_FAIL	Logical Disk Manager could not retrieve information about a legacy basic volume.
0x410000E9L LDM_E_NO_FT	There is no legacy basic volume on the disk.
0x410000EAL LDM_E_INSUFFICIENT_SPACE	There is not enough space for the configuration database on the disk.
0x410000EBL LDM_E_FT_UNHEALTHY	A legacy basic volume is unhealthy.
0x410000EDL LDM_E_NEEDS_REBOOT	The conversion will require you to restart your computer.
0x410000EEL LDM_E_BAD_ACTIVEPARTITION	An active partition is not the current active partition.
0x410000F0L LDM_E_TOO_MANY_PARTITIONS	The disk cannot be partitioned after it has been converted.
0x410000F1L LDM_E_ENCAP_PENDING	A disk conversion is already pending.
0x410000F2L LDM_E_ENCAP_FAIL	Logical Disk Manager could not convert the selected disks.
0x410000F3L LDM_E_WRONG_DISKSET_ID	The disk configuration was changed on another system.
0x410000F4L LDM_REENCAP_ABORT	Logical Disk Manager detected a previous conversion attempt failure.
0x410000F5L LDM_ENCAP_DONE	The selected disk has already been converted.
0x410000F6L LDM_FT_INCONSISTENT	The recorded configuration of legacy basic volumes before the conversion does not match the current configuration.
0x410000F7L	A foreign partition has been detected between

Return value/code	Description
LDM_MIXED_PARTITIONS	basic partitions.
0xC10000F8L LDM_E_CARVE_PARTITION	Logical Disk Manager could not create a partition for the specified volume.
0xC10000F9L LDM_E_FT_PRESENT	Logical Disk Manager does not support conversion to dynamic for disks that contain legacy basic volumes.
0xC10003E8L LDM_E_NODGINFO	Logical Disk Manager could not find the primary disk group.
0xC10003EAL LDM_E_INVALIDOBJID	The type of a cached object is not valid.
0xC10003EBL LDM_E_NORECORD	A cached object could not be found.
0xC10003EDL LDM_E_NOTVMDISK	The disk does not belong to a disk group.
0xC10003F2L LDM_E_VOLUME_IN_USE	The request cannot be completed because the volume is open or in use.
0xC10003F3L LDM_E_FORMATINPROGRESS	The volume specified cannot be formatted because the system is busy formatting another volume. Wait until that format operation is complete before continuing.
0xC10003F5L LDM_E_OBJECT_STALE	The disk and volume information in the Disk Management snap-in is out of date. To refresh the disk and volume information, press F5, or click Refresh on the Action menu.
0xC10003F6L LDM_E_BAD_FS	The operation did not complete because the file system is not compatible.
0xC10003F7L LDM_E_BAD_MEDIA	The operation did not complete because the media is not compatible.
0xC10003F8L LDM_E_NOACCESS	The operation did not complete because access is denied. Check your access permissions.
0xC10003F9L LDM_E_WRITE_PROTECTED	The operation did not complete because the media is write-protected.
0xC10003FAL LDM_E_BAD_LABEL	The operation did not complete because the label supplied is not valid.
0xC10003FBL LDM_E_CANNOT_QUICK_FORMAT	The operation did not complete because a quick format is not possible.
0xC10003FCL LDM_E_IO_ERROR	The operation did not complete because an I/O error occurred. Check the System Event Log for more information.
0xC10003FDL	The operation requires that a drive letter be

Return value/code	Description
LDM_E_NO_DRIVE_LETTER	assigned to the volume.
0xC10003FEL LDM_E_FILE_NOT_FOUND	The operation did not complete because the required file is not present.
0xC10003FFL LDM_E_CANNOT_LOAD	The restore operation did not complete because either the floppy disk does not contain the configuration information or the information is corrupted.
0xC1000400L LDM_E_MOUNTPOINT_BUSY	The drive letter could not be assigned because it is already in use.
0xC1000401L LDM_E_FORMAT_FAILED	The format did not complete successfully.
0xC1000402L LDM_E_SERVER_NOTREADY	The Logical Disk Manager Administrative Service is not yet ready to accept connections.
0xC1000403L LDM_E_CANT_PROTECT_SYSTEM	Logical Disk Manager cannot secure the system partition .
0xC1000404L LDM_E_CANT_UNPROTECT_SYSTEM	Logical Disk Manager cannot unsecure the system partition.
0xC1000405L LDM_E_FAILED_SHUTDOWN	Logical Disk Manager cannot shut down the system.
0xC1000406L LDM_E_VOL_TOO_BIG	The volume size is too big for the selected file system.
0xC1000407L LDM_E_VOL_TOO_SMALL	The volume size is too small for the selected file system.
0xC1000408L LDM_CLUSTER_SIZE_TOO_BIG	The cluster size is too big for the selected file system.
0xC1000409L LDM_CLUSTER_SIZE_TOO_SMALL	The cluster size is too small for the selected file system.
0x8100040AL LDM_W_CANTOPENLOG	Debug log file "%1" could not be opened. Debug tracing is not available.
0x8100040BL LDM_W_TIMEOUT	The request timed out and could not be completed.
0x8100040CL LDM_W_UNSUPPORTED	The requested operation is not supported.
0x0100040DL LDM_S_REBOOT_PENDING	The drive letter reassignment will not occur until the computer is restarted.
0x8100040EL LDM_W_VOL_COMPRESS_FAILED	The format succeeded, but file and folder compression are not enabled.
0xC1000411L LDM_E_VMCONFIG_LOAD_FAILED	Windows cannot load the following Logical Disk Manager configuration library: "%1.dll".

Return value/code	Description
	Logical Disk Manager might not be properly installed, or the system folder might be corrupted. Contact your system administrator for assistance.
0xC1000412L LDM_E_VOLUMEDISABLED	The requested operation is not supported on failed volumes.
0xC1000413L LDM_E_RDONLY	The requested operation cannot be completed because the media is write-protected.
0xC1000414L LDM_E_NO_DYNAMIC	Dynamic disks are not supported on this system.
0xC1000415L LDM_E_INVALID_FS	The path provided is on a file system that does not support drive paths.
0xC1000416L LDM_E_INVALID_PATH	The path cannot be used for creating a drive path because the folder does not exist or is already a drive path to some other volume.
0xC1000417L LDM_E_DIR_NOT_EMPTY	The path cannot be used for creating a drive path because the folder is not empty.
0xC1000418L LDM_E_NOT_DRIVE_PATH	Logical Disk Manager cannot delete the drive path.
0xC1000419L LDM_E_INVALID_NAME	The path is not valid and cannot be used for creating a drive path.
0xC100041BL LDM_E_MEDIA_DOESNT_SUPPORT_MOUNT_POINTS	Drive paths are not supported for removable media .
0xC100041CL LDM_E_DELETE_NOACCESS	The drive path cannot be deleted because access is denied.
0xC100041DL LDM_E_VOLUME_DISABLED	The operation did not complete because the partition or volume is not enabled. To enable the partition or volume, restart the computer.
0xC100041EL LDM_CLUSTER_COUNT_TOO_HIGH	The format operation did not complete because the cluster count is higher than expected.
0x8100041FL LDM_W_MARK_ACTIVE_FAILED_PRIMARY	The volume was repaired but the underlying partition was not marked active because another partition on the disk is already marked active. Mark the volume active to mark its underlying partitions active.
0x81000420L LDM_W_MARK_ACTIVE_FAILED_LOGICAL	The volume was repaired but the underlying partition was not marked active because it is a logical drive .
0xC1000421L LDM_E_SERVICE_DISABLED	The Logical Disk Manager Administrative Service is disabled.
0xC1000422L	Logical Disk Manager could not update the

Return value/code	Description
LDM_E_BOOTFILE	boot file for any boot partitions on the destination disk. Verify your ARC path listings in file boot.ini or through the bootcfg.exe tool.
0xC1000423L LDM_E_BAD_HARDWARE	The disk configuration operation did not complete. Check the System Event Log for more information on the error. Verify the status of your storage devices before retrying. If that does not solve the problem, close the Disk Management console, then restart Disk Management or restart the computer.
0xC1000424L LDM_E_BIOS_OFF_OR_HOTPLUG	Arcpath information for the destination disk does not exist. Either the disk was added after startup, or the SCSI BIOS is disabled on the destination disk's controller. Unable to update the boot file. Verify your arcpath listings in file boot.ini or through the bootcfg.exe tool.
0xC1000425L LDM_E_VOLUME_GROW_FAILED_FS	Failed to extend the file system for the volume.
0xC1000426L LDM_E_GPT_BOOT_MIRRORED_TO_MBR	You have mirrored your current boot volume to an master boot record (MBR) disk. You will not be able to boot from the plex on the MBR disk.
0xC1000427L LDM_E_REGION_LATENCY_RETRY	The cache has not completely updated after adding or removing partitions. Retry the operation.
0xC10007CAL LDM_E_GENERIC_ERROR	The operation did not complete. Check the System Event Log for more information on the error.
0xC10007CBL LDM_E_GENERIC_RETRY	The operation did not complete. Check the System Event Log for more information on the error. Retrying the operation may fix the problem.
0xC10007CCL LDM_E_GENERIC_RESTART	The operation did not complete. Check the System Event Log for more information on the error. Close the Disk Management console, then restart Disk Management before retrying the operation.
0xC10007CDL LDM_E_GENERIC_REBOOT	The operation did not complete. Check the System Event Log for more information on the error. Restart the computer before retrying the operation.
0xC10007CEL LDM_E_NO_VM	No dynamic disks are present.
0xC10007CFL LDM_E_INTERNALFAILURE	An internal error has occurred. Close the Disk Management console, then restart Disk Management, or restart the computer.

Return value/code	Description
0xC10007D0L LDM_E_PAGEFILE_VOLUME	This volume contains a pagefile.
0xC10007D1L LDM_E_SYSTEM_VOLUME	This volume is marked as an active (system) volume.
0xC10007D2L LDM_E_CRASHDUMP_VOLUME	This volume is configured to hold a "crashdump" file.
0xC10007D3L LDM_E_CRASHDUMP_PAGEFILE_BOOT_SYSTEM_VOLUME	The request cannot be completed because the volume is open or in use. It may be configured as a system, boot, or pagefile volume, or, to hold a "crashdump" file.

2.2.2 MAX_FS_NAME_SIZE Constant

Constant/value	Description
MAX_FS_NAME_SIZE 8	Used to define the IFILE_SYSTEM_INFO structure. It is the maximum size of a file system name, in characters, including the terminating NULL character. It is defined as a DWORD .

2.2.3 REGIONTYPE

The **REGIONTYPE** enumeration defines values for region types.

```
typedef enum _REGIONTYPE
{
    REGION_UNKNOWN = 0,
    REGION_FREE = 1,
    REGION_EXTENDED_FREE = 2,
    REGION_PRIMARY = 3,
    REGION_LOGICAL = 4,
    REGION_EXTENDED = 5,
    REGION_SUBDISK = 6,
    REGION_CDROM = 7,
    REGION_REMOVABLE = 8
} REGIONTYPE;
```

REGION_UNKNOWN: Region type is unknown.

REGION_FREE: Region resides in **free space**.

REGION_EXTENDED_FREE: Region resides in the free space of an **extended partition**.

REGION_PRIMARY: Region resides in a **primary partition**.

REGION_LOGICAL: Region resides in a **logical partition**.

REGION_EXTENDED: Region resides in an extended partition.

REGION_SUBDISK: Region resides on a subdisk.

REGION_CDROM: Region resides on a CD-ROM device.

REGION_REMOVABLE: Region resides on a device with removable media.

2.2.4 VOLUMETYPE

The **VOLUMETYPE** enumeration defines values for types of volumes.

```
typedef enum _VOLUMETYPE
{
    VOLUMETYPE_UNKNOWN = 0,
    VOLUMETYPE_PRIMARY_PARTITION = 1,
    VOLUMETYPE_LOGICAL_DRIVE = 2,
    VOLUMETYPE_FT = 3,
    VOLUMETYPE_VM = 4,
    VOLUMETYPE_CDROM = 5,
    VOLUMETYPE_REMOVABLE = 6
} VOLUMETYPE;
```

VOLUMETYPE_UNKNOWN: Volume is of an unknown type.

VOLUMETYPE_PRIMARY_PARTITION: Volume is a primary partition.

VOLUMETYPE_LOGICAL_DRIVE: Volume is a logical drive.

VOLUMETYPE_FT: Volume is a Windows NT 4.0–style **fault-tolerant** volume.

VOLUMETYPE_VM: Volume is controlled by the **volume manager**.

VOLUMETYPE_CDROM: Volume resides on a CD-ROM device.

VOLUMETYPE_REMOVABLE: Volume resides on a device with removable media.

2.2.5 VOLUMELAYOUT

The **VOLUMELAYOUT** enumeration defines values for volume layouts.

```
typedef enum _VOLUMELAYOUT
{
    VOLUMELAYOUT_UNKNOWN = 0,
    VOLUMELAYOUT_PARTITION = 1,
    VOLUMELAYOUT_SIMPLE = 2,
    VOLUMELAYOUT_SPANNED = 3,
    VOLUMELAYOUT_MIRROR = 4,
    VOLUMELAYOUT_STRIPE = 5,
    VOLUMELAYOUT_RAID5 = 6
} VOLUMELAYOUT;
```

VOLUMELAYOUT_UNKNOWN: Volume has an unknown layout.

VOLUMELAYOUT_PARTITION: Volume is a partition.

VOLUMELAYOUT_SIMPLE: Volume is a basic disk.

VOLUMELAYOUT_SPANNED: Volume spans multiple disks.

VOLUMELAYOUT_MIRROR: Volume is a mirror.

VOLUMELAYOUT_STRIPE: Volume is a striped set.

VOLUMELAYOUT_RAID5: Volume is a RAID-5 set.

2.2.6 REQSTATUS

The **REQSTATUS** enumeration defines values for the status of a request.

```
typedef enum _REQSTATUS
{
    REQ_UNKNOWN = 0,
    REQ_STARTED = 1,
    REQ_IN_PROGRESS = 2,
    REQ_COMPLETED = 3,
    REQ_ABORTED = 4,
    REQ_FAILED = 5
} REQSTATUS;
```

REQ_UNKNOWN: Request state is unknown.

REQ_STARTED: Request has started.

REQ_IN_PROGRESS: Request is in progress.

REQ_COMPLETED: Request has completed.

REQ_ABORTED: Request has terminated.

REQ_FAILED: Request has failed.

2.2.7 REGIONSTATUS

The **REGIONSTATUS** enumeration defines values for a **region's status**.

```
typedef enum _REGIONSTATUS
{
    REGIONSTATUS_UNKNOWN = 0,
    REGIONSTATUS_OK = 1,
    REGIONSTATUS_FAILED = 2,
    REGIONSTATUS_FAILING = 3,
    REGIONSTATUS_REGENERATING = 4,
    REGIONSTATUS_NEEDSRESYNC = 5
} REGIONSTATUS;
```

REGIONSTATUS_UNKNOWN: Region's status is unknown.

REGIONSTATUS_OK: Region is intact.

REGIONSTATUS_FAILED: Region failed.

REGIONSTATUS_FAILING: Region is in the process of failing.

REGIONSTATUS_REGENERATING: Region is regenerating data from the fault-tolerant check information.

REGIONSTATUS_NEEDSRESYNC: Region needs resynchronization.

2.2.8 VOLUMESTATUS

The **VOLUMESTATUS** enumeration defines values for a volume's status. For more information about redundant data and fault-tolerant volumes, see [\[MSDN-DISKMAN\]](#).

```
typedef enum _VOLUMESTATUS
{
    VOLUME_STATUS_UNKNOWN = 0,
    VOLUME_STATUS_HEALTHY = 1,
    VOLUME_STATUS_FAILED = 2,
    VOLUME_STATUS_FAILED_REDUNDANCY = 3,
    VOLUME_STATUS_FAILING = 4,
    VOLUME_STATUS_FAILING_REDUNDANCY = 5,
    VOLUME_STATUS_FAILED_REDUNDANCY_FAILING = 6,
    VOLUME_STATUS_SYNCHING = 7,
    VOLUME_STATUS_REGENERATING = 8,
    VOLUME_STATUS_INITIALIZING = 9,
    VOLUME_STATUS_FORMATTING = 10
} VOLUMESTATUS;
```

VOLUME_STATUS_UNKNOWN: Volume has an unknown status.

VOLUME_STATUS_HEALTHY: Volume is fully functional.

VOLUME_STATUS_FAILED: Volume is in a failed state.

VOLUME_STATUS_FAILED_REDUNDANCY: Volume's redundant data in a fault-tolerant volume has failed.

VOLUME_STATUS_FAILING: Volume has encountered I/O errors.

VOLUME_STATUS_FAILING_REDUNDANCY: Volume is fault-tolerant, and it encountered I/O errors.

VOLUME_STATUS_FAILED_REDUNDANCY_FAILING: Redundant data in a fault-tolerant volume has failed, and the volume encountered I/O errors in the last remaining copy of the data.

VOLUME_STATUS_SYNCHING: Volume is resynchronizing fault-tolerant data for a mirrored volume.

VOLUME_STATUS_REGENERATING: Volume is regenerating fault-tolerant data for a RAID-5 volume.

VOLUME_STATUS_INITIALIZING: Volume is initializing to volume manager control.

VOLUME_STATUS_FORMATTING: Volume is currently being formatted.

2.2.9 LdmObjectId

This type is declared as follows:

```
typedef LONGLONG LdmObjectId;
```

LdmObjectId defines a unique identifier for disk management objects such as regions, disks, and volumes. Each **LdmObjectId** MUST contain a 64-bit integer, which is unique across all disk management objects on the server.

2.2.10 VOLUME_SPEC

The **volumespec** structure specifies a new volume to create. **VOLUME_SPEC** is a typedef of this structure.

```
typedef struct volumespec {
    VOLUMETYPE type;
    VOLUME_LAYOUT layout;
    REGIONTYPE partitiontype;
    LONGLONG length;
    LONGLONG lastKnownState;
} VOLUME_SPEC;
```

type: Specifies the volume type.

layout: Specifies the volume layout.

partitiontype: Specifies the type of the underlying region, if this volume will be a partition.

length: Specifies the length of the volume in bytes. The volume length MUST always be a multiple of the disk sector size.

lastKnownState: Specifies the volume's last known **modification sequence number**.

2.2.11 VOLUME_INFO

The **VOLUME_INFO** structure provides information about a volume.

```
typedef struct volumeinfo {
    LdmObjectId id;
    VOLUMETYPE type;
    VOLUME_LAYOUT layout;
    LONGLONG length;
    LdmObjectId fsId;
    unsigned long memberCount;
    VOLUME_STATUS status;
    LONGLONG lastKnownState;
    LdmObjectId taskId;
    unsigned long vflags;
} VOLUME_INFO;
```

id: Specifies the object identifier for the volume.

type: Specifies the volume type.

layout: Specifies the volume layout.

length: Specifies the length of the volume in bytes.

fsId: Specifies the object identifier for the volume's file system, which defaults to 0 if no file system is present on the volume.

memberCount: Specifies the number of regions that compose the volume.

status: Specifies the volume status.

lastKnownState: Specifies the volume's modification sequence number.

taskId: Specifies the task identifier of the associated user request. If no request is made, the value is 0. For more information, see section [2.2.17](#).

vflags: Specifies the bitmap of volume flags. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

This field MUST be one of the following values:

Value	Meaning
VOLUME_FORMAT_IN_PROGRESS 0x00000001	Volume is currently being formatted.
VOLUME_HAS_PAGEFILE 0x00000004	Volume contains the paging file .
VOLUME_IS_BOOT_VOLUME 0x00000100	Volume contains the boot partition.
VOLUME_IS_RESTARTABLE 0x00000400	The RestartVolume method can be successfully called on this volume.
VOLUME_IS_SYSTEM_VOLUME 0x00000800	Volume contains the system directory .
VOLUME_HAS_RETAIN_PARTITION 0x00001000	Volume has an underlying partition.
VOLUME_HAD_BOOT_INI 0x00002000	Volume contained the Boot.ini file used when the operating system was last started.
VOLUME_CORRUPT 0x00004000	Volume is corrupt.
VOLUME_HAS_CRASHDUMP 0x00008000	Volume contains a crash dump file .
VOLUME_IS_CURR_BOOT_VOLUME 0x00010000	Volume is the current boot volume.
VOLUME_HAS_HIBERNATION 0x00020000	Volume contains a hibernation image .

2.2.12 DISK_SPEC

The **DISK_SPEC** structure specifies a disk for a volume modification or a creation request.


```
typedef struct diskspec {
    LdmObjectId diskId;
    LONGLONG length;
    boolean needContiguous;
    LONGLONG lastKnownState;
} DISK_SPEC;
```

diskId: Specifies the object identifier for the disk.

length: Specifies the byte length to use.

needContiguous: Boolean value that specifies if contiguous space is needed on the disk.

Value	Meaning
FALSE 0	Contiguous space is not needed on the disk.
TRUE 1	Contiguous space is needed on the disk.

lastKnownState: Last known modification sequence number of the disk.

2.2.13 REGION_SPEC

The **REGION_SPEC** structure specifies a region for partition creation and deletion.

```
typedef struct regionspec {
    LdmObjectId regionId;
    REGIONTYPE regionType;
    LdmObjectId diskId;
    LONGLONG start;
    LONGLONG length;
    LONGLONG lastKnownState;
} REGION_SPEC;
```

regionId: Specifies the object identifier for the region.

regionType: Specifies the region type.

diskId: Specifies the object identifier for the disk on which the region resides.

start: Specifies the byte offset of the region on disk.

length: Specifies the length of the region in bytes.

lastKnownState: Specifies the region's last known modification sequence number.

2.2.14 DRIVE_LETTER_INFO

The **DRIVE_LETTER_INFO** structure provides information about a drive letter. It is used for drive letter assignment and free requests, for notification of drive letter changes, and for enumeration.

```
typedef struct driveletterinfo {
    wchar_t letter;
    LdmObjectId storageId;
    boolean isUsed;
    hyper lastKnownState;
    LdmObjectId taskId;
    unsigned long dlflags;
} DRIVE_LETTER_INFO;
```

letter: Drive letter as a single case-insensitive alphabetical **Unicode** character.

storageId: Specifies the object identifier of the volume, partition, or logical drive to which the drive letter is assigned, if any.

isUsed: Boolean value that specifies if the drive letter is in use.

Value	Meaning
FALSE 0	Drive letter is free.
TRUE 1	Drive letter is in use.

lastKnownState: Modification sequence number of the drive letter.

taskId: Specifies the task identifier of the associated user request. If no request is made, the value is 0. For more information about this task identifier, see section [2.2.17](#).

dlflags: Bitmap of drive letter flags. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
DL_PENDING_REMOVAL 0x00000001	Drive letter has a removal operation pending.

2.2.15 FILE_SYSTEM_INFO

The **FILE_SYSTEM_INFO** structure provides information about a file system. This structure is used for file system enumeration, file system operations, and notification of file system changes in the configuration database. For more information about the parameters, see [\[MSDN-STC\]](#).

```
typedef struct filesysteminfo {
    LdmObjectId id;
    LdmObjectId storageId;
    LONGLONG totalAllocationUnits;
    LONGLONG availableAllocationUnits;
    unsigned long allocationUnitSize;
    unsigned long fsflags;
    hyper lastKnownState;
    LdmObjectId taskId;
    long fsType;
    int cchLabel;
```

```

    [size_is(cchLabel)] wchar_t* label;
} FILE_SYSTEM_INFO;

```

id: Specifies the object identifier for the file system.

storageId: Specifies the object identifier for the volume, partition, or logical drive associated with the file system.

totalAllocationUnits: Total number of **file allocation units** in the file system.

availableAllocationUnits: Number of available file allocation units in the file system.

allocationUnitSize: Size of a file allocation unit in bytes.

fsflags: Bitmap of **file system flags**. The value of this field is generated by combining zero or more of the applicable flags with a logical OR operation.

Value	Meaning
ENABLE_VOLUME_COMPRESSION 0x00000001	File system supports New Technology File System (NTFS) compression.

lastKnownState: File system's last known modification sequence number.

taskId: Specifies the task identifier of the associated user request. If no request is made, the value is 0. For more information about this task identifier, see section [2.2.17](#).

fsType: Type of the file system.

Value	Meaning
FSTYPE_UNKNOWN 0x00000000	File system type is unknown.
FSTYPE_NTFS 0x00000001	File system type is NTFS.
FSTYPE_FAT 0x00000002	File system type is file allocation table (FAT).
FSTYPE_FAT32 0x00000003	File system type is FAT32.
FSTYPE_CDFS 0x00000004	File system type is Compact Disk File System (CDFS) .
FSTYPE_UDF 0x00000005	File system type is Universal Disk Format (UDF) .
FSTYPE_OTHER 0x80000000	File system type is not listed.

cchLabel: Length of the label of the file system, in Unicode characters, including null termination.

label: Null terminated label of the file system. This is Unicode.

2.2.16 IFILE_SYSTEM_INFO

The **IFILE_SYSTEM_INFO** structure provides information about an installed file system. For more information, see [\[MSDN-STC\]](#).

```
typedef struct ifilesysteminfo {
    long fsType;
    WCHAR fsName[MAX_FS_NAME_SIZE];
    unsigned long fsFlags;
    unsigned long fsCompressionFlags;
    int cchLabelLimit;
    int cchLabel;
    [size_is(cchLabel)] wchar_t* iLabelChSet;
} IFILE_SYSTEM_INFO;
```

fsType: Type of the file system. This field contains one of the following values.

Value	Meaning
FSTYPE_UNKNOWN 0x00000000	File system type is unknown.
FSTYPE_NTFS 0x00000001	File system type is NTFS.
FSTYPE_FAT 0x00000002	File system type is FAT.
FSTYPE_FAT32 0x00000003	File system type is FAT32.
FSTYPE_CDFS 0x00000004	File system type is CDFS.
FSTYPE_UDF 0x00000005	File system type is UDF.
FSTYPE_OTHER 0x80000000	File system type is not listed.

fsName: Null-terminated Unicode file system name.

fsFlags: Bitmap of file system flags. The value of this field is a logical OR of zero or more of the applicable flags.

Value	Meaning
FSF_FMT_OPTION_COMPRESS 0x00000001	File system supports compression.
FSF_FMT_OPTION_LABEL 0x00000002	File system supports label specification.

Value	Meaning
FSF_MNT_POINT_SUPPORT 0x00000004	File system supports creation of mount points .
FSF_REMOVABLE_MEDIA_SUPPORT 0x00000008	File system supports creation of removable media.
FSF_FS_GROW_SUPPORT 0x00000010	File system supports the extend operation.
FSF_FS_QUICK_FORMAT_ENABLE 0x00000020	File system supports quick formatting.
FSF_FS_ALLOC_SZ_512 0x00000040	File system supports an allocation unit size of 512 bytes.
FSF_FS_ALLOC_SZ_1K 0x00000080	File system supports an allocation unit size of 1 kilobyte (KB).
FSF_FS_ALLOC_SZ_2K 0x00000100	File system supports an allocation unit size of 2 KB.
FSF_FS_ALLOC_SZ_4K 0x00000200	File system supports an allocation unit size of 4 KB.
FSF_FS_ALLOC_SZ_8K 0x00000400	File system supports an allocation unit size of 8 KB.
FSF_FS_ALLOC_SZ_16K 0x00000800	File system supports an allocation unit size of 16 KB.
FSF_FS_ALLOC_SZ_32K 0x00001000	File system supports an allocation unit size of 32 KB.
FSF_FS_ALLOC_SZ_64K 0x00002000	File system supports an allocation unit size of 64 KB.
FSF_FS_ALLOC_SZ_128K 0x00004000	File system supports an allocation unit size of 128 KB.
FSF_FS_ALLOC_SZ_256K 0x00008000	File system supports an allocation unit size of 256 KB.
FSF_FS_ALLOC_SZ_OTHER 0x00010000	File system supports any allocation unit size that the user provides.
FSF_FS_FORMAT_SUPPORTED 0x00020000	File system supports formatting.
FSF_FS_VALID_BITS 0x0003FFFF	All other bits in the bitmap MUST be ignored. The server does a bitwise AND operation with this value to clear upper-level bits that may be present but are not supported.

fsCompressionFlags: Bitmap of allocation unit sizes that are valid for compression. The value of this field is a logical 'OR' of zero or more of the applicable flags.

Value	Meaning
FSF_FS_ALLOC_SZ_1K 0x00000080	File system supports an allocation unit size of 1 KB.
FSF_FS_ALLOC_SZ_2K 0x00000100	File system supports an allocation unit size of 2 KB.
FSF_FS_ALLOC_SZ_4K 0x00000200	File system supports an allocation unit size of 4 KB.
FSF_FS_ALLOC_SZ_8K 0x00000400	File system supports an allocation unit size of 8 KB.
FSF_FS_ALLOC_SZ_16K 0x00000800	File system supports an allocation unit size of 16 KB.
FSF_FS_ALLOC_SZ_32K 0x00001000	File system supports an allocation unit size of 32 KB.
FSF_FS_ALLOC_SZ_64K 0x00002000	File system supports an allocation unit size of 64 KB.
FSF_FS_ALLOC_SZ_128K 0x00004000	File system supports an allocation unit size of 128 KB.
FSF_FS_ALLOC_SZ_256K 0x00008000	File system supports an allocation unit size of 256 KB.
FSF_FS_ALLOC_SZ_OTHER 0x00010000	File system supports any allocation unit size that the user provides.

cchLabelLimit: Maximum number of characters allowed in the file system's label.

cchLabel: Length of the iLabelChSet member in bytes.

iLabelChSet: Array of characters that are not allowed in the file system's label.

2.2.17 TASK_INFO

The **TASK_INFO** structure provides information about a task on the server.

```
typedef struct taskinfo {
    LdmObjectId id;
    LdmObjectId storageId;
    LONGLONG createTime;
    LdmObjectId clientID;
    unsigned long percentComplete;
    REQSTATUS status;
    DMPROGRESS_TYPE type;
    HRESULT error;
    unsigned long tflag;
} TASK_INFO;
```

id: Specifies the object identifier for the task.

storageId: Specifies the object identifier of the object associated with the task.

createTime: Unused. This field MUST be set to 0 by servers and ignored by clients.

clientId: Specifies the object identifier of the client that requested the task.

percentComplete: Percentage of the task completed. This field MUST be between 0 and 100, inclusive.

status: Specifies the status of the request.

type: Specifies the kind of operation referred to by the **percentComplete** member. For more information, see section [2.2.18](#).

error: The [HRESULT](#) error if the value of the **status** member is REQ_FAILED.

tflag: Unused. This field MUST be set to 0 by servers and ignored by clients.

A **TASK_INFO** structure is returned by all Disk Management methods that perform configuration operations. The **TASK_INFO** structure provides information about the task that is being performed by the server in response to the request. The **id** member of this structure identifies this task from all other tasks being performed by the server. Notifications received by the client as a task progresses can be associated with the original request by comparing the **taskId** member of the notification structure with the **id** member of this structure.

2.2.18 DMPROGRESS_TYPE

The **DMPROGRESS_TYPE** enumeration is defined as follows:

```
typedef enum _dmProgressType
{
    PROGRESS_UNKNOWN = 0,
    PROGRESS_FORMAT = 1,
    PROGRESS_SYNCHING = 2
} DMPROGRESS_TYPE;
```

PROGRESS_UNKNOWN: Unknown type of operation is in progress.

PROGRESS_FORMAT: Format operation is in progress.

PROGRESS_SYNCHING: Synchronization operation is in progress.

2.2.19 COUNTED_STRING

The **COUNTED_STRING** structure provides information about a **mounted folder**.

```
typedef struct countedstring {
    LdmObjectId sourceId;
    LdmObjectId targetId;
    int cchString;
    [size_is(cchString)] wchar_t* sstring;
} COUNTED_STRING;
```

- sourceId:** Specifies the object identifier of the source volume. The source volume has a folder to which the target volume will be mounted.
- targetId:** Specifies the object identifier of the target volume.
- cchString:** Specifies the length of the **mount path**, including null termination.
- sstring:** Null-terminated Unicode string containing the mount path of the source.

2.2.20 MERGE_OBJECT_INFO

The **MERGE_OBJECT_INFO** structure provides change information for a merge operation.

```
typedef struct mergeobjectinfo {
    DWORD type;
    DWORD flags;
    VOLUMELAYOUT layout;
    LONGLONG length;
} MERGE_OBJECT_INFO;
```

- type:** This parameter MUST be set to 0x00000001.
- flags:** Bitmap of merge flags. The value of this field is generated by combining zero or more of the applicable flags with a logical OR operation.

Value	Meaning
DSKMERGE_DELETE 0x00000001	Volume will be deleted.
DSKMERGE_DELETE_REDUNDANCY 0x00000002	Redundant data in a fault-tolerant volume will be deleted.
DSKMERGE_STALE_DATA 0x00000004	Volume contents will be stale.
DSKMERGE_RELATED 0x00000008	Volume has subdisks on merged disks.

- layout:** Value from the [VOLUMELAYOUT](#) enumeration that indicates the volume's new layout.
- length:** Volume's new size in bytes.

2.3 IVolumeClient Interface

2.3.1 IVolumeClient Data Types

2.3.1.1 PARTITION_OS2_BOOT Constant

Constant/value	Description
PARTITION_OS2_BOOT 0xa	This constant is a value for the mbr.partitionType member of the REGION_INFO_EX structure. It is defined as an unsigned long .<7>

2.3.1.2 DISK_INFO

The **DISK_INFO** structure provides information about a disk.

```
typedef struct diskinfo {
    LdmObjectId id;
    LONGLONG length;
    LONGLONG freeBytes;
    unsigned long bytesPerTrack;
    unsigned long bytesPerCylinder;
    unsigned long bytesPerSector;
    unsigned long regionCount;
    unsigned long dflags;
    unsigned long deviceType;
    unsigned long deviceState;
    unsigned long busType;
    unsigned long attributes;
    boolean isUpgradeable;
    int portNumber;
    int targetNumber;
    int lunNumber;
    LONGLONG lastKnownState;
    LdmObjectId taskId;
    int cchName;
    int cchVendor;
    int cchDgid;
    int cchAdapterName;
    int cchDgName;
    [sizeis(cchName)] wchar_t* name;
    [sizeis(cchVendor)] wchar_t* vendor;
    [sizeis(cchDgid)] byte* dgid;
    [sizeis(cchAdapterName)] wchar_t* adapterName;
    [sizeis(cchDgName)] wchar_t* dgName;
} DISK_INFO;
```

id: Specifies the object identifier of the disk.

length: Size of the disk in bytes.

freeBytes: Number of unallocated bytes on the disk.

bytesPerTrack: Size of a disk **track** in bytes.

bytesPerCylinder: Size of a disk **cylinder** in bytes.

bytesPerSector: Size of a disk sector in bytes.

regionCount: Total number of regions on the disk.

dflags: **Disk type** of the disk.

Value	Meaning
DISK_AUDIO_CD 0x00000001	Disk is an audio CD.
DISK_NEC98 0x00000002	This value is obsolete and MUST NOT be used.

deviceType: Device type of the disk. This field contains one of the following values.

Value	Meaning
DEVICETYPE_UNKNOWN 0x00000000	Device is of an unknown type.
DEVICETYPE_VM 0x00000001	Device is a dynamic disk.
DEVICETYPE_REMOVABLE 0x00000002	Device uses removable media.
DEVICETYPE_CDROM 0x00000003	Device is a CD-ROM.
DEVICETYPE_FDISK 0x00000004	Device is a basic disk .
DEVICETYPE_DVD 0x00000005	Device is a DVD.

deviceState: Device state of the disk. The value of this field is generated by combining zero or more of the applicable flags with a logical OR operation. Valid combinations are device-type dependent.

Value	Meaning
DEVICESTATE_UNKNOWN 0x00000000	Disk is in an unknown state.
DEVICESTATE_HEALTHY 0x00000001	Disk is fully functional.
DEVICESTATE_NO_MEDIA 0x00000002	Disk has no media.
DEVICESTATE_NOSIG 0x00000004	Disk has an invalid signature .
DEVICESTATE_BAD 0x00000008	Disk was deleted or experienced an install or hardware problem.
DEVICESTATE_NOT_READY	Disk is not ready yet.

Value	Meaning
0x00000010	
DEVICESTATE_MISSING 0x00000020	Disk is no longer available.
DEVICESTATE_OFFLINE 0x00000040	Disk is offline.
DEVICESTATE_FAILING 0x00000080	Disk experienced a physical input/output (I/O) error.
DEVICESTATE_IMPORT_FAILED 0x00000100	Disk belongs to a group whose import failed.
DEVICESTATE_UNCLAIMED 0x00000200	Disk belongs to a foreign disk group.

busType: Type of **bus** on which the disk resides. This field contains one of the following values.

Value	Meaning
BUSTYPE_UNKNOWN 0x00000000	Bus type is unknown.
BUSTYPE_IDE 0x00000001	Disk resides on an integrated drive electronics (IDE) bus .
BUSTYPE_SCSI 0x00000002	Disk resides on a small computer system interface (SCSI) bus .
BUSTYPE_FIBRE 0x00000003	Disk resides on a Fiber Channel bus.
BUSTYPE_USB 0x00000004	Disk resides on a universal serial bus (USB) .
BUSTYPE_SSA 0x00000005	Disk resides on a Serial Storage Architecture (SSA) Bus .
BUSTYPE_1394 0x00000006	Disk resides on an Institute of Electronics and Electrical Engineers (IEEE) 1394 bus.

attributes: Bitmap of disk attributes. The value of this field is generated by combining zero or more of the applicable flags defined in the following table with a logical OR operation.

Value	Meaning
DEVICEATTR_NONE 0x00000000	Disk has no attributes.
DEVICEATTR_RDONLY 0x00000001	Disk is read-only.
DEVICEATTR_NTMS 0x00000002	This value is obsolete.

isUpgradeable: Boolean value that indicates if the disk can be converted to a dynamic disk. Will be true if the disk is basic, healthy, and has 512 byte sectors.

Value	Meaning
FALSE 0	Disk cannot be converted to a dynamic disk.
TRUE 1	Disk can be encapsulated or converted to a dynamic disk.

portNumber: **SCSI port number** of the disk, if the bus reports this information.

targetNumber: SCSI target identifier of the disk, if the bus reports this information.

lunNumber: **SCSI logical unit number (LUN)** of the disk, if the bus reports this information.

lastKnownState: Modification sequence number of the disk.

taskId: The task identifier of the associated user request. If no request is made, the value is 0. For more information about this task identifier, see section [2.2.17](#).

cchName: Length of the **hard disk physical name**, in Unicode characters, including null termination.

cchVendor: Length of the disk's vendor name, in Unicode characters, including null termination.

cchDgid: Length of the disk's group identification handle, in **ASCII** characters, including null termination.

cchAdapterName: Length of the disk's adapter name, in Unicode characters, including null termination.

cchDgName: Length of the disk's group name, in Unicode characters, including null termination.

name: Null-terminated physical device name of the hard disk, in the format '\\device\\Harddisk1'. This is Unicode.

vendor: Null-terminated name of the hard disk vendor. This is the disk vendor's disk model name. This is Unicode.

dgid: Specifies the object identifier of the disk's disk group. This is ASCII.

adapterName: Null-terminated name of the **disk adapter** as returned by the disk adapter firmware. Example: 'Adaptec AHA-2940U2W - Ultra2 SCSI'. This is Unicode.

dgName: Null-terminated name for the disk's disk group, if the disk is dynamic. Only dynamic disks have an associated disk group. Basic disks do not. This is Unicode.

2.3.1.3 REGION_INFO

The **REGION_INFO** structure provides information about a region.

```
typedef struct regioninfo {  
    LdmObjectId id;  
    LdmObjectId diskId;  
    LdmObjectId volId;
```

```

LdmObjectId fsId;
LONGLONG start;
LONGLONG length;
REGIONTYPE regionType;
unsigned long partitionType;
boolean isActive;
REGIONSTATUS status;
hyper lastKnownState;
LdmObjectId taskId;
unsigned long rflags;
unsigned long currentPartitionNumber;
} REGION_INFO;

```

id: Specifies the region's object identifier.

diskId: Specifies the object identifier of the disk on which the region resides.

volId: Specifies the object identifier of the volume on the region, if any. The value of this field is nonzero if it is valid.

fsId: Specifies the object identifier of the file system on the region, if any. The value of this field is nonzero if it is valid.

start: Byte offset of the region on the disk.

length: Length of the region in bytes.

regionType: Value from the [REGIONTYPE](#) enumeration that indicates the region type.

partitionType: Type of the partition on the region. For a list of allowable partition types, see [\[MSDN-PARTITIONINFO\]](#).

isActive: Boolean value that indicates if the region is an active partition.

Value	Meaning
FALSE 0	Region is an inactive partition.
TRUE 1	Region is an active partition.

status: Value from the [REGIONSTATUS](#) enumeration that indicates the region's status.

lastKnownState: Modification sequence number of the region.

taskId: This [LdmObjectId](#) is the task identifier of the associated user request. If no request is made, the value is 0. For more information about this task identifier, see section [2.2.17](#).

rflags: Bitmap of **region flags**. The value of this field is generated by combining zero or more of the applicable flags with a logical OR operation.

Value	Meaning
REGION_FORMAT_IN_PROGRESS	Region is currently being formatted.

Value	Meaning
0x00000001	
REGION_IS_SYSTEM_PARTITION 0x00000002	Region contains the system directory. The system directory has the operating system installed to it. This is not necessarily the "active" partition that contains the boot loader file .
REGION_HAS_PAGEFILE 0x00000004	Region contains the paging file.
REGION_HAD_BOOT_INI 0x00000040	Boot file was located in this region when the operating system was last started. This is the "active" partition that contains the boot loader file.

currentPartitionNumber: Number of the partition on the region, if any. For more information, see [\[MSDN-PARTITIONINFO\]](#).

2.4 IVolumeClient2 Interface

2.4.1 IVolumeClient2 Data Types

No additional data types are defined by this interface.

2.5 IVolumeClient3 Interface

2.5.1 IVolumeClient3 Data Types

2.5.1.1 PARTITIONSTYLE

The **PARTITIONSTYLE** enumeration defines the style of a partition.

```
typedef enum _PARTITIONSTYLE
{
    PARTITIONSTYLE_UNKNOWN = 0,
    PARTITIONSTYLE_MBR = 1,
    PARTITIONSTYLE_GPT = 2
} PARTITIONSTYLE;
```

PARTITIONSTYLE_UNKNOWN: Partition is of an unknown style.

PARTITIONSTYLE_MBR: Partition is of the master boot record (MBR) style.

PARTITIONSTYLE_GPT: Partition is of the GUID partition table (GPT) style.

2.5.1.2 DISK_INFO_EX

The **DISK_INFO_EX** structure provides information about a disk.

```
typedef struct diskinfoex {
    LdmObjectId id;
    LONGLONG length;
    LONGLONG freeBytes;
    unsigned long bytesPerTrack;
```

```

unsigned long bytesPerCylinder;
unsigned long bytesPerSector;
unsigned long regionCount;
unsigned long dflags;
unsigned long deviceType;
unsigned long deviceState;
unsigned long busType;
unsigned long attributes;
unsigned long maxPartitionCount ;
boolean isUpgradeable;
boolean maySwitchStyle;
PARTITIONSTYLE partitionStyle;
[switch_is(partitionStyle)] union {
    [case(PARTITIONSTYLE_MBR)]
        struct {
            unsigned long signature;
        } mbr;
    [case(PARTITIONSTYLE_GPT)]
        struct {
            GUID diskId;
        } gpt;
    [default]
        ;
};
int portNumber;
int targetNumber;
int lunNumber;
LONGLONG lastKnownState;
LdmObjectId taskId;
int cchName;
int cchVendor;
int cchDgid;
int cchAdapterName;
int cchDgName;
int cchDevInstId;
[size_is(cchName)] wchar_t* name;
[size_is(cchVendor)] wchar_t* vendor;
[size_is(cchDgid)] byte* dgid;
[size_is(cchAdapterName)] wchar_t* adapterName;
[size_is(cchDgName)] wchar_t* dgName;
[size_is(cchDevInstId)] wchar_t* devInstId;
} DISK_INFO_EX;

```

id: Specifies the object identifier of the disk.

length: Size of the disk in bytes.

freeBytes: Number of unallocated bytes on the disk.

bytesPerTrack: Size of a disk track in bytes.

bytesPerCylinder: Size of a disk cylinder in bytes.

bytesPerSector: Size of a disk sector in bytes.

regionCount: Total number of regions on the disk.

dflags: Disk type of the disk. The value of this field is generated by combining zero or more of the applicable flags with a logical OR operation.

Value	Meaning
DISK_AUDIO_CD 0x00000001	Disk is an audio CD.
DISK_NEC98 0x00000002	This value is obsolete and MUST NOT be returned.
DISK_FORMATTABLE_DVD 0x00000004	Disk is a DVD that can be formatted.
DISK_MEMORY_STICK 0x00000008	Disk is a memory stick.
DISK_NTFS_NOT_SUPPORTED 0x00000010	Disk does not support being formatted as NTFS.

deviceType: Device type of the disk.

Value	Meaning
DEVICETYPE_UNKNOWN 0x00000000	Device is of an unknown type.
DEVICETYPE_VM 0x00000001	Device is a dynamic disk.
DEVICETYPE_REMOVABLE 0x00000002	Device uses removable media.
DEVICETYPE_CDROM 0x00000003	Device is a CD-ROM.
DEVICETYPE_FDISK 0x00000004	Device is a basic disk.
DEVICETYPE_DVD 0x00000005	Device is a DVD.

deviceState: Device state of the disk.

Value	Meaning
DEVICESTATE_UNKNOWN 0x00000000	Disk is in an unknown state.
DEVICESTATE_HEALTHY 0x00000001	Disk is fully functional.
DEVICESTATE_NO_MEDIA 0x00000002	Disk has no media.
DEVICESTATE_NOSIG 0x00000004	Disk has an invalid signature.

Value	Meaning
DEVICESTATE_BAD 0x00000008	Disk experienced a geometry failure.
DEVICESTATE_NOT_READY 0x00000010	Disk is not ready yet.
DEVICESTATE_MISSING 0x00000020	Disk is no longer available.
DEVICESTATE_OFFLINE 0x00000040	Disk is offline.
DEVICESTATE_FAILING 0x00000080	Disk experienced a physical I/O error.
DEVICESTATE_IMPORT_FAILED 0x00000100	Disk belongs to a group whose import failed. See Disk Group Import .
DEVICESTATE_UNCLAIMED 0x00000200	Disk belongs to a foreign disk group.

busType: Type of bus on which the disk resides.

Value	Meaning
BUSTYPE_UNKNOWN 0x00000000	Bus type is unknown.
BUSTYPE_IDE 0x00000001	Disk resides on an integrated drive electronics (IDE) bus.
BUSTYPE_SCSI 0x00000002	Disk resides on a small computer system interface (SCSI) bus.
BUSTYPE_FIBRE 0x00000003	Disk resides on a Fiber Channel bus.
BUSTYPE_USB 0x00000004	Disk resides on a universal serial bus (USB).
BUSTYPE_SSA 0x00000005	Disk resides on a Serial Storage Architecture (SSA) bus.
BUSTYPE_1394 0x00000006	Disk resides on an Institute of Electronics and Electrical Engineers (IEEE) 1394 bus.

attributes: Bitmap of disk attributes.

Value	Meaning
DEVICEATTR_NONE 0x00000000	Disk has no attributes.
DEVICEATTR_RDONLY 0x00000001	Disk is read-only.

Value	Meaning
DEVICEATTR_NTMS 0x00000002	This value is obsolete.

maxPartitionCount : Maximum number of partitions on the disk.

isUpgradeable: Boolean value that indicates if the disk can be converted to a dynamic disk.
True if the disk is basic, healthy, and has 512-byte sectors.

Value	Meaning
FALSE 0	Disk cannot be encapsulated or converted to a dynamic disk.
TRUE 1	Disk can be encapsulated or converted to a dynamic disk.

maySwitchStyle: Boolean value that indicates if the disk's partition style can be changed from MBR to GPT, or, changed from GPT to MBR.

Value	Meaning
FALSE 0	Partition style of the disk cannot be changed.
TRUE 1	Partition style of the disk can be changed between master boot record (MBR) and GUID partition table (GPT).

partitionStyle: Value from the [PARTITIONSTYLE](#) enumeration that indicates the disk's partitioning style.

: A union that contains either a **signature** or a **diskId**, depending on the value of **partitionStyle**:

signature: Signature of the disk. The **disk signature** is not guaranteed to be unique across machines. [<8>](#)

diskId: [GUID](#) of the disk. [<9>](#)

portNumber: SCSI port number of the disk.

targetNumber: SCSI target identifier of the disk.

lunNumber: SCSI logical unit number (LUN) of the disk.

lastKnownState: Modification sequence number of the disk.

taskId: The task identifier of the associated user request. If no request is made, the value is 0.

cchName: Length of the hard disk's physical name, including null termination.

cchVendor: Length of the disk's vendor name, including null termination.

cchDgid: Length of the disk's group identification handle, including null termination.

cchAdapterName: Length of the disk's adapter name, including null termination.

cchDgName: Length of the disk's group name, including null termination.

cchDevInstId: Length of the disk's device instance path, including null termination.

name: Null-terminated physical name of the hard disk. For example: '\\device\\Harddisk1'.

vendor: Null-terminated name of the hard disk vendor. This is the disk vendor's disk model name. For example: "SEAGATE ST34573N SCSI Disk Device".

dgid: Specifies the object identifier of the disk's disk group.

adapterName: Null-terminated name of the disk adapter. For example: "Adaptec AHA-2940U2W - Ultra2 SCSI".

dgName: Null-terminated name for the disk's disk group, if the disk is dynamic.

devInstId: Null-terminated device instance path of the disk with the backslashes replaced by "#", "\\?" prepended to the beginning, and the Pnp disk class GUID appended to the end. For example: "\\?\\ide#diskwdc_wd1600jd-75hbb008.02d08#5&15c8d966&0&0.0.0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}".

2.5.1.3 REGION_INFO_EX

The **REGION_INFO_EX** structure provides information about a region.

```
typedef struct regioninfoex {
    LdmObjectId id;
    LdmObjectId diskId;
    LdmObjectId volId;
    LdmObjectId fsId;
    LONGLONG start;
    LONGLONG length;
    REGIONTYPE regionType;
    PARTITIONSTYLE partitionStyle;
    [switch is(partitionStyle)] union {
        [case(PARTITIONSTYLE_MBR)]
        struct {
            unsigned long partitionType;
            boolean isActive;
        } mbr;
        [case(PARTITIONSTYLE_GPT)]
        struct {
            GUID partitionType;
            GUID partitionId;
            ULONGLONG attributes;
        } gpt;
        [default]
        ;
    };
    REGIONSTATUS status;
    hyper lastKnownState;
    LdmObjectId taskId;
    unsigned long rflags;
    unsigned long currentPartitionNumber;
    int cchName;
    [size_is(cchName)] wchar_t* name;
} REGION_INFO_EX;
```

id: Specifies the region's object identifier.

diskId: Specifies the object identifier of the disk on which the region resides.

volId: Specifies the object identifier of the volume on the region, if any.

fsId: Specifies the object identifier of the file system on the region, if any.

start: Byte offset of the region on the disk.

length: Length of the region in bytes.

regionType: Value from the [REGIONTYPE](#) enumeration that indicates the region type.

partitionStyle: Value from the [PARTITIONSTYLE](#) enumeration that indicates the region's partitioning style.

: A union that contains either a **partitionType** of type [ULONG](#) and an **isActive**, or a **partitionType** of type [GUID](#), a **partitionId**, and an **attributes**, depending on the value of **partitionStyle**:

partitionType: Windows NT 4.0, Windows NT 3.5, Windows NT 3.1, and Windows NT 3.51 partition style for the region. For more information, see [\[MSDN-PARTITIONINFO\]](#).

isActive: Boolean value that indicates if the partition is active. The partition MUST be marked as active for the BIOS to start from the partition on x86 and x64 platforms.

Value	Meaning
FALSE 0	Partition is not active.
TRUE 1	Partition is active.

partitionType: Windows NT partition style for the disk. For more information, see [\[MSDN-PARTITIONINFO\]](#).

partitionId: A **GUID** that uniquely identifies a partition on a disk.

attributes: Bitmap of partition flags.[<10>](#)

status: Value from the [REGIONSTATUS](#) enumeration that indicates the region's status.

lastKnownState: Modification sequence number of the region.

taskId: This [LdmObjectId](#) is the task identifier of the associated user request. If no request is made, the value MUST be 0.

rflags: Bitmap of region flags. The value of this field is generated by combining zero or more of the applicable flags with a logical OR operation.

Value	Meaning
REGION_FORMAT_IN_PROGRESS 0x00000001	Region is currently being formatted.
REGION_IS_SYSTEM_PARTITION 0x00000002	Region contains the system directory. The system directory has the operating system installed on it. This is not necessarily the "active" partition that contains the boot loader file.
REGION_HAS_PAGEFILE 0x00000004	Region contains the paging file.
REGION_HAD_BOOT_INI 0x00000040	Boot.ini file was located in this region when the operating system was last started. This is the "active" partition that contains the boot loader file.
REGION_HIDDEN 0x00040000	This region is part of a volume that is not accessible through any user-available path names. <11>

currentPartitionNumber: Number of the partition on the region, if any. For more information, see [\[MSDN-PARTITIONINFO\]](#).

cchName: Length of the region's name, including null termination.

name: Null-terminated name of the region.

2.6 IVolumeClient4 Interface

The IVolumeClient4 interface is implemented by servers to provide additional disk management support on top of the [IVolumeClient](#), [IVolumeClient2](#), and [IVolumeClient3](#) interfaces.

2.6.1 IVolumeClient4 Data Types

No additional data types are defined by this interface.

2.7 IDMRemoteServer Interface

The IDMRemoteServer interface is implemented by servers to support activation of servers on remote machines.

This DCOM interface inherits the IUnknown interface. Method opnum field values start with 3; opnum values 0–2 represent the IUnknown_QueryInterface, IUnknown_AddRef, and IUnknown_Release methods, respectively, as specified in [\[MS-DCOM\]](#).

2.7.1 IDMRemoteServer Data Types

No additional data types are defined by this interface.

2.8 IDMNotify Interface

The IDMNotify interface is implemented by the client to receive change notifications from a remote disk management server.

This DCOM interface inherits the IUnknown interface. Method opnum field values start with 3; opnum values 0–2 represent the IUnknown_QueryInterface, IUnknown_AddRef, and IUnknown_Release methods, respectively, as specified in [\[MS-DCOM\]](#).

2.8.1 IDMNotify Data Types

2.8.1.1 DMNOTIFY_INFO_TYPE

The **DMNOTIFY_INFO_TYPE** enumeration defines the type of object described by an ObjectsChanged call.

```
typedef enum _dmNotifyInfoType
{
    DMNOTIFY_UNKNOWN_INFO = 0,
    DMNOTIFY_DISK_INFO = 1,
    DMNOTIFY_VOLUME_INFO = 2,
    DMNOTIFY_REGION_INFO = 3,
    DMNOTIFY_TASK_INFO = 4,
    DMNOTIFY_DL_INFO = 5,
    DMNOTIFY_FS_INFO = 6,
    DMNOTIFY_SYSTEM_INFO = 7
} DMNOTIFY_INFO_TYPE;
```

DMNOTIFY_UNKNOWN_INFO: Object is of an unknown type.

DMNOTIFY_DISK_INFO: Object is a disk.

DMNOTIFY_VOLUME_INFO: Object is a volume.

DMNOTIFY_REGION_INFO: Object is a region.

DMNOTIFY_TASK_INFO: Object is a task.

DMNOTIFY_DL_INFO: Object is a drive letter.

DMNOTIFY_FS_INFO: Object is a file system.

DMNOTIFY_SYSTEM_INFO: Object is the Disk Management system.

2.8.1.2 LDMACTION

The **LDMACTION** enumeration defines the type of action described by an ObjectsChanged call.

```
typedef enum _LDMACTION
{
    LDMACTION_UNKNOWN = 0,
    LDMACTION_CREATED = 1,
    LDMACTION_DELETED = 2,
    LDMACTION_MODIFIED = 3,
    LDMACTION_FAILED = 4
} LDMACTION;
```

LDMACTION_UNKNOWN: Object underwent an unknown type of change.

LDMACTION_CREATED: Object was created.

LDMACTION_DELETED: Object was deleted.

LDMACTION_MODIFIED: Object was modified.

LDMACTION_FAILED: Object failed.

3 Protocol Details

The following sections specify details of the Disk Management Remote Protocol, including abstract data models, interface method syntax, and message processing rules.

3.1 Client Role Details

3.1.1 Abstract Data Model

No abstract data model is required.

3.1.2 Timers

No timers are required.

3.1.3 Initialization

A client SHOULD choose to use one of the following sets of interfaces when communicating with a Disk Management Remote Protocol server, grouped by the functionality that they offer.

- [IDMNotify](#), [IVolumeClient](#), [IVolumeClient2](#)
- [IDMNotify](#), [IVolumeClient3](#)
- [IDMNotify](#), [IVolumeClient3](#), [IVolumeClient4](#)

The client negotiates for a given set of server functionality by specifying the UUID corresponding to the wanted RPC interface when binding to the server, as specified in section [1.9](#).

The client MAY choose to use the [IDMRemoteServer](#) interface to create an instance of a Disk Management Remote Protocol remote server.

A client initializes by creating an RPC binding handle to the [IVolumeClient3](#) interface. If the [IVolumeClient3](#) interface is not advertised, or fails, the client MAY fallback to creating an RPC binding handle to the [IVolumeClient](#) interface. A description of how to get a client-side RPC binding handle for an [IVolumeClient](#) or [IVolumeClient3](#) interface is as specified in [\[MS-DCOM\]](#) section 3.1.4.

When the client has obtained the [IVolumeClient](#) or [IVolumeClient3](#) interface, the client MUST invoke the [Initialize](#) method on the interface.

When the client has called [IVolumeClient::Initialize](#) or [IVolumeClient3::Initialize](#), the client MAY begin making calls against the server.

If the client has obtained binding to the [IVolumeClient3](#) interface, it can also create an RPC binding handle to the [IVolumeClient4](#) interface to call its methods. If the client has obtained binding to the [IVolumeClient](#) interface, it can create an RPC binding handle to the [IVolumeClient2](#) interface to call its methods.

3.1.4 Message Processing and Sequencing Rules

The Message Processing Events and Sequencing Rules interface defines the following methods.

Methods in RPC Opnum Order

Method	Description
<u>IVolumeClient::EnumDisks</u>	Opnum: 3
<u>IVolumeClient::EnumDiskRegions</u>	Opnum: 4
<u>IVolumeClient::CreatePartition</u>	Opnum: 5
<u>IVolumeClient::CreatePartitionAssignAndFormat</u>	Opnum: 6
<u>IVolumeClient::CreatePartitionAssignAndFormatEx</u>	Opnum: 7
<u>IVolumeClient::DeletePartition</u>	Opnum: 8
<u>IVolumeClient::WriteSignature</u>	Opnum: 9
<u>IVolumeClient::MarkActivePartition</u>	Opnum: 10
<u>IVolumeClient::Eject</u>	Opnum: 11
<u>IVolumeClient::FTEnumVolumes</u>	Opnum: 13
<u>IVolumeClient::FTEnumLogicalDiskMembers</u>	Opnum: 14
<u>IVolumeClient::FTDeleteVolume</u>	Opnum: 15
<u>IVolumeClient::FTBreakMirror</u>	Opnum: 16
<u>IVolumeClient::FTResyncMirror</u>	Opnum: 17
<u>IVolumeClient::FTRegenerateParityStripe</u>	Opnum: 18
<u>IVolumeClient::FTReplaceMirrorPartition</u>	Opnum: 19
<u>IVolumeClient::FTReplaceParityStripePartition</u>	Opnum: 20
<u>IVolumeClient::EnumDriveLetters</u>	Opnum: 21
<u>IVolumeClient::AssignDriveLetter</u>	Opnum: 22
<u>IVolumeClient::FreeDriveLetter</u>	Opnum: 23
<u>IVolumeClient::EnumLocalFileSystems</u>	Opnum: 24
<u>IVolumeClient::GetInstalledFileSystems</u>	Opnum: 25
<u>IVolumeClient::Format</u>	Opnum: 26
<u>IVolumeClient::EnumVolumes</u>	Opnum: 28
<u>IVolumeClient::EnumVolumeMembers</u>	Opnum: 29
<u>IVolumeClient::CreateVolume</u>	Opnum: 30
<u>IVolumeClient::CreateVolumeAssignAndFormat</u>	Opnum: 31
<u>IVolumeClient::CreateVolumeAssignAndFormatEx</u>	Opnum: 32
<u>IVolumeClient::GetVolumeMountName</u>	Opnum: 33

Method	Description
<u>IVolumeClient::GrowVolume</u>	Opnum: 34
<u>IVolumeClient::DeleteVolume</u>	Opnum: 35
<u>IVolumeClient::AddMirror</u>	Opnum: 36
<u>IVolumeClient::RemoveMirror</u>	Opnum: 37
<u>IVolumeClient::SplitMirror</u>	Opnum: 38
<u>IVolumeClient::InitializeDisk</u>	Opnum: 39
<u>IVolumeClient::UninitializeDisk</u>	Opnum: 40
<u>IVolumeClient::ReConnectDisk</u>	Opnum: 41
<u>IVolumeClient::ImportDiskGroup</u>	Opnum: 43
<u>IVolumeClient::DiskMergeQuery</u>	Opnum: 44
<u>IVolumeClient::DiskMerge</u>	Opnum: 45
<u>IVolumeClient::ReAttachDisk</u>	Opnum: 47
<u>IVolumeClient::ReplaceDisk</u>	Opnum: 48
<u>IVolumeClient::ReplaceRaid5Column</u>	Opnum: 51
<u>IVolumeClient::RestartVolume</u>	Opnum: 52
<u>IVolumeClient::GetEncapsulateDiskInfo</u>	Opnum: 53
<u>IVolumeClient::EncapsulateDisk</u>	Opnum: 54
<u>IVolumeClient::QueryChangePartitionNumbers</u>	Opnum: 55
<u>IVolumeClient::DeletePartitionNumberInfoFromRegistry</u>	Opnum: 56
<u>IVolumeClient::SetDontShow</u>	Opnum: 57
<u>IVolumeClient::GetDontShow</u>	Opnum: 58
<u>IVolumeClient::EnumTasks</u>	Opnum: 67
<u>IVolumeClient::GetTaskDetail</u>	Opnum: 68
<u>IVolumeClient::AbortTask</u>	Opnum: 69
<u>IVolumeClient::HrGetErrorData</u>	Opnum: 70
<u>IVolumeClient::Initialize</u>	Opnum: 71
<u>IVolumeClient::Uninitialize</u>	Opnum: 72
<u>IVolumeClient::Refresh</u>	Opnum: 73
<u>IVolumeClient::RescanDisks</u>	Opnum: 74

Method	Description
<u>IVolumeClient::RefreshFileSys</u>	Opnum: 75
<u>IVolumeClient::SecureSystemPartition</u>	Opnum: 76
<u>IVolumeClient::ShutDownSystem</u>	Opnum: 77
<u>IVolumeClient::EnumAccessPath</u>	Opnum: 78
<u>IVolumeClient::EnumAccessPathForVolume</u>	Opnum: 79
<u>IVolumeClient::AddAccessPath</u>	Opnum: 80
<u>IVolumeClient::DeleteAccessPath</u>	Opnum: 81
<u>IVolumeClient2::GetMaxAdjustedFreeSpace</u>	Opnum: 3
<u>IVolumeClient3::EnumDisksEx</u>	Opnum: 3
<u>IVolumeClient3::EnumDiskRegionsEx</u>	Opnum: 4
<u>IVolumeClient3::CreatePartition</u>	Opnum: 5
<u>IVolumeClient3::CreatePartitionAssignAndFormat</u>	Opnum: 6
<u>IVolumeClient3::CreatePartitionAssignAndFormatEx</u>	Opnum: 7
<u>IVolumeClient3::DeletePartition</u>	Opnum: 8
<u>IVolumeClient3::InitializeDiskStyle</u>	Opnum: 9
<u>IVolumeClient3::MarkActivePartition</u>	Opnum: 10
<u>IVolumeClient3::Eject</u>	Opnum: 11
<u>IVolumeClient3::FTEnumVolumes</u>	Opnum: 13
<u>IVolumeClient3::FTEnumLogicalDiskMembers</u>	Opnum: 14
<u>IVolumeClient3::FTDeleteVolume</u>	Opnum: 15
<u>IVolumeClient3::FTBreakMirror</u>	Opnum: 16
<u>IVolumeClient3::FTResyncMirror</u>	Opnum: 17
<u>IVolumeClient3::FTRegenerateParityStripe</u>	Opnum: 18
<u>IVolumeClient3::FTReplaceMirrorPartition</u>	Opnum: 19
<u>IVolumeClient3::FTReplaceParityStripePartition</u>	Opnum: 20
<u>IVolumeClient3::EnumDriveLetters</u>	Opnum: 21
<u>IVolumeClient3::AssignDriveLetter</u>	Opnum: 22
<u>IVolumeClient3::FreeDriveLetter</u>	Opnum: 23
<u>IVolumeClient3::EnumLocalFileSystems</u>	Opnum: 24

Method	Description
<u>IVolumeClient3::GetInstalledFileSystems</u>	Opnum: 25
<u>IVolumeClient3::Format</u>	Opnum: 26
<u>IVolumeClient3::EnumVolumes</u>	Opnum: 27
<u>IVolumeClient3::EnumVolumeMembers</u>	Opnum: 28
<u>IVolumeClient3::CreateVolume</u>	Opnum: 29
<u>IVolumeClient3::CreateVolumeAssignAndFormat</u>	Opnum: 30
<u>IVolumeClient3::CreateVolumeAssignAndFormatEx</u>	Opnum: 31
<u>IVolumeClient3::GetVolumeMountName</u>	Opnum: 32
<u>IVolumeClient3::GrowVolume</u>	Opnum: 33
<u>IVolumeClient3::DeleteVolume</u>	Opnum: 34
<u>IVolumeClient3::CreatePartitionsForVolume</u>	Opnum: 35
<u>IVolumeClient3::DeletePartitionsForVolume</u>	Opnum: 36
<u>IVolumeClient3::GetMaxAdjustedFreeSpace</u>	Opnum: 37
<u>IVolumeClient3::AddMirror</u>	Opnum: 38
<u>IVolumeClient3::RemoveMirror</u>	Opnum: 39
<u>IVolumeClient3::SplitMirror</u>	Opnum: 40
<u>IVolumeClient3::InitializeDiskEx</u>	Opnum: 41
<u>IVolumeClient3::UninitializeDisk</u>	Opnum: 42
<u>IVolumeClient3::ReConnectDisk</u>	Opnum: 43
<u>IVolumeClient3::ImportDiskGroup</u>	Opnum: 44
<u>IVolumeClient3::DiskMergeQuery</u>	Opnum: 45
<u>IVolumeClient3::DiskMerge</u>	Opnum: 46
<u>IVolumeClient3::ReAttachDisk</u>	Opnum: 47
<u>IVolumeClient3::ReplaceRaid5Column</u>	Opnum: 48
<u>IVolumeClient3::RestartVolume</u>	Opnum: 49
<u>IVolumeClient3::GetEncapsulateDiskInfoEx</u>	Opnum: 50
<u>IVolumeClient3::EncapsulateDiskEx</u>	Opnum: 51
<u>IVolumeClient3::QueryChangePartitionNumbers</u>	Opnum: 52
<u>IVolumeClient3::DeletePartitionNumberInfoFromRegistry</u>	Opnum: 53

Method	Description
<u>IVolumeClient3::SetDontShow</u>	Opnum: 54
<u>IVolumeClient3::GetDontShow</u>	Opnum: 55
<u>IVolumeClient3::EnumTasks</u>	Opnum: 64
<u>IVolumeClient3::GetTaskDetail</u>	Opnum: 65
<u>IVolumeClient3::AbortTask</u>	Opnum: 66
<u>IVolumeClient3::HrGetErrorData</u>	Opnum: 67
<u>IVolumeClient3::Initialize</u>	Opnum: 68
<u>IVolumeClient3::Uninitialize</u>	Opnum: 69
<u>IVolumeClient3::Refresh</u>	Opnum: 70
<u>IVolumeClient3::RescanDisks</u>	Opnum: 71
<u>IVolumeClient3::RefreshFileSys</u>	Opnum: 72
<u>IVolumeClient3::SecureSystemPartition</u>	Opnum: 73
<u>IVolumeClient3::ShutDownSystem</u>	Opnum: 74
<u>IVolumeClient3::EnumAccessPath</u>	Opnum: 75
<u>IVolumeClient3::EnumAccessPathForVolume</u>	Opnum: 76
<u>IVolumeClient3::AddAccessPath</u>	Opnum: 77
<u>IVolumeClient3::DeleteAccessPath</u>	Opnum: 78
<u>IVolumeClient4::RefreshEx</u>	Opnum: 3
<u>IVolumeClient4::GetVolumeDeviceName</u>	Opnum: 4
<u>IDMRemoteServer::CreateRemoteObject</u>	Opnum: 3
<u>IDMNotify::ObjectsChanged</u>	Opnum: 3

The server MUST implement all above methods of the [IVolumeClient](#), [IVolumeClient2](#), [IVolumeClient3](#), and [IVolumeClient4](#) interfaces.

The server MAY implement all methods of the [IDMRemoteServer](#) interface.

The client SHOULD implement all methods of the [IDMNotify](#) interface.

Note Gaps in the opnum numbering sequence represent opnums that MUST NOT be used over the wire.

[<12>](#)

For all above return results:

If the return code is not an error, the client SHOULD assume that all output parameters are present and valid.

Exceptions Thrown: This protocol does not throw any exceptions beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#), or the operating system.

3.1.4.1 Higher-Layer Triggered Events

All method invocations are triggered by higher-layer events, such as commands issued within administrative and diagnostic applications. Details of method invocations are in the following sections.

3.1.4.1.1 Common Details

3.1.4.1.1.1 Methods with Prerequisites

Some method calls require no prerequisite calls against the server and simply query for information or pass in parameters constructed by the client. This type of method is not discussed further in the client section of this document. For more information, see section [3.2.4.4](#).

Other calls are required to be made in sequence and are listed here. The prerequisite call is to an object enumeration method that retrieves information about a specific set of storage objects, such as volumes or disks. Information returned by the object enumeration method is then used to supply input parameters for subsequent calls. Calls with such prerequisites are grouped by storage object type in the following sections.

3.1.4.1.1.2 Parameters to [IVolumeClient](#) and [IVolumeClient3](#)

In the client processing outlined below, when calling methods in [IVolumeClient3](#) rather than [IVolumeClient](#), references to the [DISK_INFO](#) structure MUST be replaced with [DISK_INFO_EX](#); and references to the [REGION_INFO](#) structure MUST be replaced with [REGION_INFO_EX](#). Similarly, a call to [EnumDisks](#) MUST be replaced with a call to [EnumDisksEx](#); and a call to [EnumDiskRegions](#) MUST be replaced with a call to [EnumDiskRegionsEx](#).

In the call order descriptions, if any prerequisite calls fail, the call order cannot proceed.

3.1.4.1.1.3 Relationships Between Storage Objects

Regions and Volumes: Given a [REGION_INFO](#) that describes a region, a client can map the region to its volume by obtaining a list of [VOLUME_INFO](#) structures (as returned from a call to [EnumVolumes](#)) and matching the [REGION_INFO::volId](#) member to the [VOLUME_INFO::id](#) member of an entry in the list of volumes.

Regions and Disks: Given a [REGION_INFO](#) that describes a region, a client can map the region to its disk by obtaining a list of [DISK_INFO](#) structures (as returned from a call to [EnumDisks](#)) and matching the [REGION_INFO::diskId](#) member to the [DISK_INFO::id](#) member of an entry in the list of disks.

Regions and File systems: Given a [REGION_INFO](#) that describes a region, a client can map the region to its file system by obtaining a list of [FILE_SYSTEM_INFO](#) structures (as returned from a call to [EnumLocalFileSystems](#)) and matching the [REGION_INFO::fsId](#) member to the [FILE_SYSTEM_INFO::id](#) member of an entry in the list of file systems.

Volumes and File systems: Given a [VOLUME_INFO](#) structure that describes a volume, a client can map the volume to a file system by obtaining a list of [FILE_SYSTEM_INFO](#) structures (as returned from a call to [EnumLocalFileSystems](#)) and matching the [VOLUME_INFO::fsId](#) member to the [FILE_SYSTEM_INFO::id](#) member of an entry in the list of file systems.

Volumes and Tasks: Given a [TASK_INFO](#) structure returned by a call to perform an operation on a volume (such as Format or Grow Volume), the TASK_INFO::storageId member will map to the VOLUME_INFO::id member of the volume. Conversely, the VOLUME_INFO::taskId member maps to the TASK_INFO::id member.

Volumes and Drive Letters: Given a **VOLUME_INFO** structure that describes a volume, a client can map the volume to its drive letter by obtaining a list of [DRIVE_LETTER_INFO](#) structures (as returned from a call to [EnumDriveLetters](#)) and matching the VOLUME_INFO::id member to the DRIVE_LETTER_INFO::storageId member of an entry in the list of file systems.

Volume Members and Regions: Given a list of [LdmObjectIds](#) that identify **volume members** (as returned from a call to [EnumVolumeMembers](#)), a client can map the list of **LdmObjectIds** to the **REGION_INFO** structures of volume members as follows:

- Obtain a list of **DISK_INFO** structures (as returned from a call to [EnumDisks](#)) for all disks.
- For each entry in the list of **DISK_INFO** structures, obtain a list of the **REGION_INFO** structures (as returned from a call to [EnumDiskRegions](#)) for all regions on the disk.
- For each entry in the list of **REGION_INFO** structures, match the REGION_INFO::id member structure to an entry in the list of **LdmObjectId**.

3.1.4.1.2 Drive Letters

AssignDriveLetter: Prior to invoking [AssignDriveLetter](#), the client MUST invoke [FTEnumVolumes](#), [EnumDiskRegions](#), or [EnumVolumes](#) to retrieve the volume ID and the volume's last known state. The client MUST pass these returned values as the *storageId* and *storageLastKnownState* input parameters to the **AssignDriveLetter** method. **EnumDiskRegions** returns these values as the **REGION_INFO::volId** and **REGION_INFO::lastKnownState** structure members. **FTEnumVolumes** and **EnumVolumes** return these parameters as the **VOLUME_INFO::id** and **VOLUME_INFO::lastKnownState** structure members.

Prior to invoking **AssignDriveLetter**, the client MUST also invoke [EnumDriveLetters](#). **EnumDriveLetters** will return the drive letter's last known state as **DRIVE_LETTER_INFO::lastKnownState**. The client MUST pass this returned value as the *letterLastKnownState* input parameter to the **AssignDriveLetter** method. **EnumDriveLetters** also returns the status of the drive letter (in use or free) as the **DRIVE_LETTER_INFO::inUse** structure member; and this value can be used to determine if the drive letter is already in use by some other volume.

FreeDriveLetter: The client MUST use the call sequence description above for **AssignDriveLetter**, except that in the final step it MUST use [FreeDriveLetter](#) to free the drive letter, instead of **AssignDriveLetter**.

3.1.4.1.3 File Systems

AddAccessPath: Prior to invoking [AddAccessPath](#), the client MUST invoke [FTEnumVolumes](#), [EnumDiskRegions](#), or [EnumVolumes](#) to retrieve the volume ID. The client MUST pass this returned value as the *targetId* input parameter to the [AddAccessPath](#). **EnumDiskRegions** returns this value as the **REGION_INFO::volId** structure member. **FTEnumVolumes** and **EnumVolumes** return this parameter as the **VOLUME_INFO::id** structure members.

DeleteAccessPath: The client MUST use the call sequence description above for **AddAccessPath**, except that in the final step it MUST use [DeleteAccessPath](#), instead of **AddAccessPath**.

Format: Prior to invoking Format, the client MUST invoke **FTEnumVolumes**, **EnumDiskRegions**, or **EnumVolumes** to retrieve the volume ID and the volume's last known state. The client MUST pass these returned values as the *storageId* and *storageLastKnownState* input parameters to the Format method. **EnumDiskRegions** returns these values as the **REGION_INFO::volId** and **REGION_INFO::lastKnownState** structure members. **FTEnumVolumes** and **EnumVolumes** return these parameters as the **VOLUME_INFO::id** and **VOLUME_INFO::lastKnownState** structure members.

Prior to invoking Format, the client MUST also invoke [GetInstalledFileSystems](#) to retrieve the available file system types. The requested file system type, as specified by the **FILE_SYSTEM_INFO::fsType** input parameter for the call to Format, MUST be one of the types returned by **GetInstalledFileSystems**. **GetInstalledFileSystems** returns this information as the **FILE_SYSTEM_INFO::fsType** structure member.

When calling Format, the client MUST initialize the **FILE_SYSTEM_INFO::fsType**, **FILE_SYSTEM_INFO::label**, **FILE_SYSTEM_INFO::fsflags**, and **FILE_SYSTEM_INFO::allocationUnitSize** fields in the [FILE_SYSTEM_INFO](#) structure. The other fields are not used for this call.

GetVolumeMountName: Prior to invoking [GetVolumeMountName](#), the client MUST invoke **FTEnumVolumes**, **EnumDiskRegions**, or **EnumVolumes** to retrieve the volume ID. The client MUST pass this returned value as the *volumeId* input parameter to the **GetVolumeMountName** method. **EnumDiskRegions** returns this value as the **REGION_INFO::volId** structure member. **FTEnumVolumes** and **EnumVolumes** return this parameter as the **VOLUME_INFO::id** structure members.

EnumAccessPathForVolume: The client MUST use the call sequence description above for [GetVolumeMountName](#), except that in the final step it MUST use [EnumAccessPathForVolume](#), instead of **GetVolumeMountName**.

3.1.4.1.4 Disks

DiskMerge: For call sequencing related to the *dgid* and *cchDgid* input parameters, see [ImportDiskGroup](#). For call sequencing related to the *numDisks* and *diskList* input parameters, see [ImportDiskGroup](#).

Prior to invoking [DiskMerge](#), the client MUST invoke [DiskMergeQuery](#) to retrieve the disk group's last known state, an array of disk IDs, and a count of the disks in the array. The client MUST pass these returned values as the *merge_config_tid*, *merge_dm_rids*, and *numRids* input parameters to the **DiskMerge** method. **DiskMergeQuery** returns these values as the *merge_config_tid*, *merge_dm_rids*, and *numRids* output parameters.

DiskMergeQuery: Prior to invoking **DiskMergeQuery**, the client MUST invoke [EnumDisks](#) to retrieve the disk group's ID and the disk group's count of characters in the disk group ID. The client MUST pass these returned values as the *dgid* and *cchDgid* input parameters to the **DiskMergeQuery** method. **EnumDisks** returns these values as the **DISK_INFO::dgid** and **DISK_INFO::cchDgid** structure members.

Prior to invoking **DiskMergeQuery**, the client MUST invoke **EnumDisks** to retrieve the number of disks and a list of the disk IDs. The client MUST pass these returned values as the *numDisks* and *diskList* input parameters to the **DiskMergeQuery** method. **EnumDisks** returns an array of [DISK_INFO](#) structures; and the client MUST construct the list of disk IDs and the count of disks in the list, using this array of **DISK_INFO** structures.

Eject: Prior to invoking [Eject](#), the client MUST invoke **EnumDisks** to retrieve the disk ID and the disk's last known state. The client MUST pass these returned values as the *diskId* and

diskLastKnownState input parameters to the **Eject** method. **EnumDisks** returns these values as the **DISK_INFO::id** and **DISK_INFO::lastKnownState** structure members.

EncapsulateDisk: Prior to invoking [EncapsulateDisk](#), the client MUST invoke [GetEncapsulateDiskInfo](#) to retrieve the lists of disks that will be converted to dynamic disks, as well as the lists of volumes and regions on these disks. [GetEncapsulateDiskInfo](#) also returns the count of items in each of these lists. The client MUST pass these returned values as the *affectedDiskList*, *affectedVolumeList*, *affectedRegionList*, *affectedDiskCount*, *affectedVolumeCount*, and *affectedRegionCount* in the input parameters to the **EncapsulateDisk** method. These values are returned by [GetEncapsulateDiskInfo](#) as the *affectedDiskList*, *affectedVolumeList*, *affectedRegionList*, *affectedDiskCount*, *affectedVolumeCount*, and *affectedRegionCount* output parameters.

EncapsulateDiskEx: Prior to invoking [EncapsulateDiskEx](#), the client MUST invoke [GetEncapsulateDiskInfoEx](#) to retrieve the lists of disks that will be converted to dynamic, as well as the lists of volumes and regions on these disks. [GetEncapsulateDiskInfoEx](#) also returns the count of items in each of these lists. The client MUST pass these returned values as the *affectedDiskList*, *affectedVolumeList*, *affectedRegionList*, *affectedDiskCount*, *affectedVolumeCount*, and *affectedRegionCount* in the input parameters to the **EncapsulateDiskEx** method. These values are returned by [GetEncapsulateDiskInfoEx](#) as the *affectedDiskList*, *affectedVolumeList*, *affectedRegionList*, *affectedDiskCount*, *affectedVolumeCount*, and *affectedRegionCount* output parameters.

EnumDiskRegions: Prior to invoking [EnumDiskRegions](#), the client MUST invoke [EnumDisks](#) to retrieve the disk ID. The client MUST pass this returned value as the *diskId* input parameter to the **EnumDiskRegions** method. **EnumDisks** returns this value as the **DISK_INFO::id** structure member.

EnumDiskRegionsEx: Prior to invoking **EnumDiskRegionsEx**, the client MUST invoke **EnumDisksEx** to retrieve the disk ID. The client MUST pass this returned value as the *diskId* input parameter to the **EnumDiskRegionsEx** method. **EnumDisksEx** returns this value as the **DISK_INFO_EX::id** structure member.

GetEncapsulateDiskInfo: Prior to invoking [GetEncapsulateDiskInfo](#), the client MUST invoke **EnumDisks** to retrieve the disk ID and the disk's last known state. The client MUST pass these returned values as the [DISK_SPEC](#) structure's **ID** and **lastKnownState** members in the input parameter to the [GetEncapsulateDiskInfo](#) method. These values are returned by **EnumDisks** as **DISK_INFO::id** and **DISK_INFO::lastKnownState**.

GetEncapsulateDiskInfoEx: Prior to invoking [GetEncapsulateDiskInfoEx](#), the client MUST invoke **EnumDisksEx** to retrieve the disk ID and the disk's last known state. The client MUST pass these returned values as the [DISK_SPEC](#) structure's **ID** and **DISK_INFO::lastKnownState** members in the input parameter to the [GetEncapsulateDiskInfoEx](#) method. These values are returned by **EnumDisksEx** as **DISK_INFO_EX::id** and **DISK_INFO_EX::lastKnownState**.

GetMaxAdjustedFreeSpace: The client MUST use the call sequence description above for **EnumDiskRegions**, except that in the final step it MUST use [GetMaxAdjustedFreeSpace](#), instead of **EnumDiskRegions**.

ImportDiskGroup: Prior to invoking **ImportDiskGroup**, the client MUST invoke **EnumDisks** to retrieve the disk group's ID and the disk group's count of characters in the disk group ID. The client MUST pass these returned values as the *dgid* and *cchDgid* input parameters to the **ImportDiskGroup** method. **EnumDisks** returns these values as the **DISK_INFO::dgid** and **DISK_INFO::cchDgid** structure members.

InitializeDisk: Prior to invoking [InitializeDisk](#), the client MUST invoke **EnumDisks** to retrieve the disk ID and the disk's last known state. The client MUST pass these returned values as the *diskId* and *diskLastKnownState* input parameters to the **InitializeDisk** method. **EnumDisks** returns these values as the **DISK_INFO::id** and **DISK_INFO::lastKnownState** structure members.

InitializeDiskEx: The client MUST use the call sequence description below for [InitializeDiskStyle](#), except that in the final step it MUST use **InitializeDiskEx**, instead of **InitializeDiskStyle**.

InitializeDiskStyle: Prior to invoking **InitializeDiskStyle**, the client MUST invoke **EnumDisksEx** to retrieve the disk ID, the disk's last known state, and the partition style. The client MUST pass these returned values as the *diskId*, *style*, and *diskLastKnownState* input parameters to the **InitializeDiskStyle** method. **EnumDisksEx** returns these values as the **DISK_INFO_EX::id**, **DISK_INFO_EX::lastKnownState**, and **DISK_INFO_EX::partitionStyle** structure members.

QueryChangePartitionNumbers: This call SHOULD be made after calling the **EncapsulateDisk** method to determine if the conversion of basic disks to dynamic disks has caused a boot partition number to change. [<13>](#)

ReAttachDisk: The client MUST use the preceding call sequence description for [WriteSignature](#), except that in the final step it MUST use [ReAttachDisk](#) instead of **WriteSignature**.

ReConnectDisk: Prior to invoking [ReConnectDisk](#), the client MUST invoke **EnumDisks** to retrieve the disk ID. The client MUST pass this returned value as the *diskId* input parameter to the **ReConnectDisk** method. **EnumDisks** returns this value as the **DISK_INFO::id** structure member.

ReplaceDisk: Prior to invoking [ReplaceDisk](#), the client MUST invoke **EnumDisks** to retrieve disk IDs and last known states for both the replacement disk and the disk being replaced. The client MUST pass the returned disk IDs as the *newDiskId* and *oldDiskId* input parameters to the **ReplaceDisk** method. **EnumDisks** returns these values as the **DISK_INFO::id** structure member. The client MUST also pass the returned last known state for the replacement disk and the disk being replaced as the *newDiskLastKnownState* and *oldDiskLastKnownState* input parameters to **ReplaceDisk**. **EnumDisks** returns these values as the **DISK_INFO::lastKnownState** structure member.

UninitializeDisk: Prior to invoking [UninitializeDisk](#), the client MUST invoke **EnumDisks** to retrieve the disk id and the disk's last known state. The client MUST pass these returned values as the *diskId* and *diskLastKnownState* input parameters to the **UninitializeDisk** method. **EnumDisks** returns these values as the **DISK_INFO::id** and **DISK_INFO::lastKnownState** structure members.

WriteSignature: Prior to invoking **WriteSignature**, the client MUST invoke **EnumDisks** to retrieve the disk ID and the disk's last known state. The client MUST pass these returned values as the *diskId* and *diskLastKnownState* input parameters to the **WriteSignature** method. **EnumDisks** returns these values as the **DISK_INFO::id** and **DISK_INFO::lastKnownState** structure members.

3.1.4.1.5 Partitions

CreatePartition: Prior to invoking [CreatePartition](#), the client MUST invoke [EnumDiskRegions](#) to retrieve the region ID, the region type, the disk ID, and the region's last known state. The client MUST pass these returned values as the [REGION_SPEC](#) structure's **regionId**, **regionType**, **diskId**, and **lastKnownState** members in the input parameter to the **CreatePartition** method. These values are returned by **EnumDiskRegions** as **REGION_INFO::id**, **REGION_INFO::regionType**, **REGION_INFO::diskId**, and **REGION_INFO::lastKnownState**.

CreatePartitionAssignAndFormat: For call sequencing related to the **REGION_SPEC** input parameter, see section [3.2.4.4.1.3](#). For call sequencing related to the *letterLastKnownState* input parameter, see section [3.2.4.4.1.19](#). For call sequencing related to the *FILE_SYSTEM_INFO::fsType* input parameter, see section [3.2.4.4.1.23](#).

CreatePartitionAssignAndFormatEx: For call sequencing related to the **REGION_SPEC** input parameter, see [CreatePartition](#). For call sequencing related to the *letterLastKnownState* input parameter, see section [3.2.4.4.1.19](#). For call sequencing related to the *FILE_SYSTEM_INFO::fsType* input parameter, see section [3.2.4.4.1.23](#).

When calling [CreatePartitionAssignAndFormatEx](#), the client MUST initialize the **FILE_SYSTEM_INFO::fsType**, **FILE_SYSTEM_INFO::label**, **FILE_SYSTEM_INFO::fsflags**, and **FILE_SYSTEM_INFO::allocationUnitSize** fields in the **FILE_SYSTEM_INFO** structure. The other fields are not used for this call.

DeletePartition: The client MUST use the call sequence description above for [CreatePartition](#), except that in the final step it MUST use [DeletePartition](#), instead of [CreatePartition](#).

EnumVolumeMembers: The client MUST use the call sequence description above for [GetVolumeMountName](#), except that in the final step it MUST use [EnumVolumeMembers](#), instead of [GetVolumeMountName](#).

MarkActivePartition: Prior to invoking [MarkActivePartition](#), the client MUST invoke **EnumDiskRegions** to retrieve the region id and the region's last known state. The client MUST pass these returned values as the *regionId* and *regionLastKnownState* input parameters to the **MarkActivePartition** method. **EnumDiskRegions** returns these values as the **REGION_INFO::id** and **REGION_INFO::lastKnownState** structure members.

3.1.4.1.6 Volumes

AddMirror: The client MUST use the preceding call sequence description for [DeleteVolume](#) to retrieve the volume ID and volume's last known state input parameters, except that in the final step it MUST not call [DeleteVolume](#). The client MUST use the preceding call sequence description for [CreateVolume](#) to retrieve the **DISK_SPEC** input parameter, except that in the final step it MUST not call **CreateVolume**. The client MUST pass these input parameters to the [AddMirror](#) method.

The client MUST set the *diskNumber* input parameter to 0.

Checking if a disk has enough free space to host the new copy of the **volume data** can be done by examining the free regions returned by the [EnumDiskRegions](#) method.

CreatePartitionsForVolume: The client MUST use the preceding call sequence description for [DeleteVolume](#), except that in the final step it MUST use [CreatePartitionsForVolume](#), instead of [DeleteVolume](#).

CreateVolume: Prior to invoking **CreateVolume**, the client MUST invoke [EnumDisks](#) to retrieve the disk ID and the disk's last known state. The client MUST pass these returned values as the **DISK_SPEC** structure's ID and *lastKnownState* members in the input parameter to the [CreateVolume](#) method. These values are returned by **EnumDisks** as *DISK_INFO::id* and *DISK_INFO::lastKnownState*.

The client MUST call **EnumDiskRegions** method with the disk ID as input parameter and examine the free regions returned to see if the disk had enough free space to host the new volume. For the *VOLUME_SPEC* input parameter: The **lastKnownState**, **type**, and **partitionType** members are ignored. These parameters MUST be set to 0.

CreateVolumeAssignAndFormat: The client MUST use the preceding call sequence description for **CreateVolume** to retrieve the **DISK_SPEC** input parameter, except that in the final step it MUST not call **CreateVolume**. If the client does not want to assign a drive letter, the letter parameter MUST be a two-byte NULL character or unicode SPACE character. If the clients wants to assign a drive letter, the client MUST use the preceding call sequence for [AssignDriveLetter](#) to retrieve the *letterLastKnownState* input parameter, except that in the final step it MUST not call **AssignDriveLetter**. The client MUST use the preceding call sequence for [Format](#) to retrieve the *FILE_SYSTEM_INFO::fsType* input parameter, except that in the final step it MUST not call **Format**. The client MUST pass these input parameters to the [CreateVolumeAssignAndFormat](#) method.

CreateVolumeAssignAndFormatEx: The client MUST use the preceding call sequence description for **CreateVolume** to retrieve the **DISK_SPEC** input parameter, except that in the final step it MUST not call **CreateVolume**. If the client does not want to assign a drive letter, the letter parameter MUST be a two-byte NULL character or unicode SPACE character. If the clients wants to assign a drive letter, the client MUST use the preceding call sequence for **AssignDriveLetter** to retrieve the *letterLastKnownState* input parameter, except that in the final step it MUST not call **AssignDriveLetter**. The client MUST use the preceding call sequence for **Format** to retrieve the *FILE_SYSTEM_INFO::fsType* input parameter, except that in the final step it MUST not call **Format**. The client MUST pass these input parameters to the [CreateVolumeAssignAndFormatEx](#) method.

DeletePartitionsForVolume: The client MUST use the preceding call sequence description for **DeleteVolume**, except that in the final step it MUST use [DeletePartitionsForVolume](#), instead of **DeleteVolume**.

DeleteVolume: Prior to invoking **DeleteVolume**, the client MUST invoke **EnumDiskRegions** or [EnumVolumes](#) to retrieve the volume ID and the volume's last known state. The client MUST pass these returned values as the *volumeId* and *volumeLastKnownState* input parameters to the **DeleteVolume** method. **EnumDiskRegions** returns these values as the **REGION_INFO::volId** and **REGION_INFO::lastKnownState** structure members. **EnumVolumes** returns these parameters as the **VOLUME_INFO::id** and **VOLUME_INFO::lastKnownState** structure members.

FTBreakMirror: The client MUST use the preceding call sequence description for [FTDeleteVolume](#), except that in the final step it MUST use [FTBreakMirror](#), instead of **FTDeleteVolume**.

FTDeleteVolume: Prior to invoking **FTDeleteVolume**, the client MUST invoke [FTEnumVolumes](#) or **EnumDiskRegions** to retrieve the volume ID and the volume's last known state. The client MUST pass these returned values as the *volumeId* and *volumeLastKnownState* input parameters to the **FTDeleteVolume** method. **EnumDiskRegions** returns these values as the **REGION_INFO::volId** and **REGION_INFO::lastKnownState** structure members. **FTEnumVolumes** returns these parameters as the **VOLUME_INFO::id** and **VOLUME_INFO::lastKnownState** structure members.

FTEnumLogicalDiskMembers: Prior to invoking [FTEnumLogicalDiskMembers](#), the client MUST invoke **FTEnumVolumes** or **EnumDiskRegions** to retrieve the volume ID. The client MUST pass this returned value as the *volumeId* input parameter to the **FTEnumLogicalDiskMembers** method. **EnumDiskRegions** returns this value as the **REGION_INFO::volId** structure member; **FTEnumVolumes** returns this parameter as the **VOLUME_INFO::id** structure members.

FTRegenerateParityStripe: The client MUST use the preceding call sequence description for **FTDeleteVolume**, except that in the final step it MUST use [FTRegenerateParityStripe](#), instead of **FTDeleteVolume**.

FTReplaceMirrorPartition: Prior to invoking [FTReplaceMirrorPartition](#), the client MUST invoke **FTDeleteVolume** or **EnumDiskRegions** to retrieve the volume ID and the volume's last known state. The client MUST pass these returned values as the *volumeId* and *volumeLastKnownState* input parameters to the **FTReplaceMirrorPartition** method. **EnumDiskRegions** returns these

values as the **REGION_INFO::volId** and **REGION_INFO::lastKnownState** structure members; **FTEnumVolumes** returns these parameters as the **VOLUME_INFO::id** and **VOLUME_INFO::lastKnownState** structure members.

Prior to invoking **FTReplaceMirrorPartition**, the client MUST also invoke **EnumDiskRegions** to retrieve the region ID and the region's last known state. The client MUST pass these returned values as the *newRegionId* and *newRegionLastKnownState* input parameters to the **FTReplaceMirrorPartition** method. **EnumDiskRegions** returns these values as the **REGION_INFO::volId** and **REGION_INFO::lastKnownState** structure members.

Note The client MUST pass 0 for the *oldMemberId* and *oldMemberLastKnownState* parameters since they are not used or implemented.

FTReplaceParityStripePartition: The client MUST use the preceding call sequence description for **FTReplaceMirrorPartition**, except that in the final step it MUST use [FTReplaceParityStripePartition](#), instead of **FTReplaceMirrorPartition**.

FTResyncMirror: The client MUST use the preceding call sequence description for **FTDeleteVolume**, except that in the final step it MUST use [FTResyncMirror](#), instead of **FTDeleteVolume**.

GetVolumeDeviceName: The client MUST use the preceding call sequence description for [GetVolumeMountName](#), except that in the final step it MUST use [GetVolumeDeviceName](#), instead of **GetVolumeMountName**.

GrowVolume: The client MUST use the preceding call sequence description for **GetVolumeMountName** to retrieve the volume ID input parameter, except that in the final step it MUST not call **GetVolumeMountName**. The client MUST use the preceding call sequence description for **CreateVolume** to retrieve the **DISK_SPEC** input parameter, except that in the final step it MUST not call **CreateVolume**.

Prior to invoking [GrowVolume](#), the client MUST invoke **EnumVolumes** to retrieve the volume layout and the volume's last known state. The client MUST pass these returned values as the **VOLUME_SPEC** structure's layout and **lastKnownState** members in the input parameter to the **GrowVolume** method. These values are returned by **EnumVolumes** as **VOLUME_INFO::layout** and **VOLUME_INFO::lastKnownState**.

RemoveMirror: The client MUST use the preceding call sequence description for **DeleteVolume** to retrieve the volume ID and volume's last known state input parameters, except that in the final step it MUST not call **DeleteVolume**. The client MUST use the preceding call sequence description for **WriteSignature** to retrieve the *diskId* and *diskLastKnownState* input parameters, except that in the final step it MUST not call **WriteSignature**. The client MUST pass these input parameters to the [RemoveMirror](#) method.

ReplaceRaid5Column: The client MUST use the preceding call sequence description for **DeleteVolume**, except that in the final step it MUST use [ReplaceRaid5Column](#), instead of **DeleteVolume**.

Prior to invoking **ReplaceRaid5Column**, the client MUST invoke [EnumDisks](#) to retrieve the replacement disk id and the disk's last known state. The client MUST pass these returned values as the *newDiskId* and *diskLastKnownState* input parameters to the **ReplaceRaid5Column** method. **EnumDisks** returns these values as the **DISK_INFO::id** and **DISK_INFO::lastKnownState** structure members.

RestartVolume: The [RestartVolume](#) MUST use the preceding call sequence description for **DeleteVolume**, except that in the final step it MUST use **RestartVolume**, instead of **DeleteVolume**.

SplitMirror: For call sequencing related to the *volumeId* and *volumeLastKnownState* input parameters, see *DeleteVolume*. For call sequencing related to the *diskId* and *diskLastKnownState* input parameters, see *WriteSignature*. For call sequencing related to the *letterLastKnownState* input parameter, see *AssignDriveLetter*. The client MUST use the preceding call sequence description for **DeleteVolume** to retrieve the volume ID and volume's last known state input parameters, except that in the final step it MUST not call *DeleteVolume*. The client MUST use the preceding call sequence description for *WriteSignature* to retrieve the *diskId* and *diskLastKnownState* input parameters, except that in the final step it MUST not call *WriteSignature*. If the client does not want to assign a drive letter, the letter parameter MUST be a two-byte NULL character or unicode SPACE character. If the clients wants to assign a drive letter, the client MUST use the call sequence description for [AssignDriveLetter](#) to retrieve the *letterLastKnownState* input parameter, except that in the final step it MUST not call **AssignDriveLetter**. To force the split, the client MUST set the *TASK_INFO::error* parameter to LDM_E_VOLUME_IN_USE. The client MUST pass these input parameters to the [SplitMirror](#) method.

3.1.4.1.7 Tasks

GetTaskDetail: Prior to invoking [GetTaskDetail](#), the client MUST invoke [EnumTasks](#) to retrieve the task ID. The client MUST pass this returned value as the ID input parameters to the **GetTaskDetail** method. **EnumTasks** returns this value as the **TASK_INFO::id** structure member.

AbortTask: For call sequencing related to the ID input parameter, see **GetTaskDetail**. The client MUST use the preceding call sequence description for **GetTaskDetail**, except that in the final step it MUST use [AbortTask](#), instead of **GetTaskDetail**.

3.1.4.1.8 Loss of Connection

In the event of loss of connection to the server, the client MUST not use any server state that was returned in prior method invocations when the connection is reestablished, including [LdmObjectId](#), *LastKnownState* returned by the server, and MUST clean up all local resources that were allocated to the connection.

3.1.4.2 Processing Server Replies to Method Calls

Upon receiving a reply from the server in response to a method call, the client MUST validate the return code. Return codes from all method calls are [HRESULTs](#). If the **HRESULT** indicates success, the client MAY assume that all output parameters are present and valid.

Certain calls are required to be performed in sequence. For example, where method A is a prerequisite call for method B, the client MUST pass output parameters from method A as input parameters to method B, as specified in section [3.1.4](#). The client MUST retain the output parameters from method A until method B has been called.

3.1.4.3 Processing Notifications Sent from the Server to the Client

The client MAY choose to implement the [IDMNotify](#) interface to receive notification from the server of changes to the storage objects on the server. Notifications are sent to the client for storage object creation, deletion, and modification. The client MAY choose to take some other action based on these notifications. The client MAY also choose to ignore notifications from the server. [<14>](#)

Notifications related to storage object modification indicate a state change, such as when a region's status has changed from *REGIONSTATUS_OK* to *REGIONSTATUS_FAILED*, or when a volume's length changes as the result of a call to *GrowVolume*.

Notifications containing a **TASK_INFO** structure indicate the status of a method call. Method calls that operate on storage objects return a **TASK_INFO** structure. This structure contains a status field. The status field of a **TASK_INFO** structure contains one of the following values: REQ_UNKNOWN, REQ_STARTED, REQ_IN_PROGRESS, REQ_COMPLETED, REQ_ABORTED, or REQ_FAILED. When a method call returns a success code as its **HRESULT** return code, the client MAY check the **TASK_INFO** structure's status field to determine the state of the server processing associated with the method call. Only if the **TASK_INFO::status** field is REQ_STARTED or REQ_IN_PROGRESS will the client receive any further notifications regarding this operation.

If the value of the **TASK_INFO::status** field is REQ_UNKNOWN, the client MUST assume that the server has encountered a catastrophic error. In this case, the client MUST assume that no further task notifications will be received.

If the value of the **TASK_INFO::status** field is REQ_COMPLETED, REQ_ABORTED, or REQ_FAILED, the client will not receive any further task notifications. If the value of the status field is REQ_COMPLETED, this indicates that the server processing completed without errors. If the value of the status field is REQ_ABORTED, this indicates that the server processing was interrupted and did not complete successfully. If the value of the status field is REQ_FAILED, this indicates that the server processing failed and did not complete successfully. In this case, the client inspects the **TASK_INFO::error** field.

If the value of the **TASK_INFO::status** field is REQ_STARTED or REQ_IN_PROGRESS, the client MUST assume that it will receive a **TASK_INFO** notification with the status field set to REQ_COMPLETED, REQ_ABORTED, or REQ_FAILED when the server has completed its processing for the operation. The client MUST NOT assume the server's processing is complete until a task notification with one of these status values has been received.

If the network connection fails, it MUST be reestablished and all server state MUST be refreshed by the client. In this case, in progress task information is lost to the client.

The client MAY receive one or more task notifications with the **TASK_INFO::status** value set to REQ_IN_PROGRESS. Notifications with this task status value are sent to indicate server progress while processing an operation request. If the client receives a notification with this **TASK_INFO::status** value, the client MAY inspect the **TASK_INFO::percentComplete** field to determine task progress.

The client maps task notifications received to a given method call based on the **TASK_INFO::id** field. This field is unique per method call.

For a full description of the IDMNotify interface and the ObjectsChanged method, see section [3.1.4.4.1](#).

3.1.4.4 Protocol Message Details

3.1.4.4.1 IDMNotify Methods

Methods in RPC Opnum Order

Method	Description
IDMNotify::ObjectsChanged	Opnum: 3

3.1.4.4.1.1 IDMNotify::ObjectsChanged (Opnum 3)

The **ObjectsChanged** method notifies the client of object changes.

```

HRESULT ObjectsChanged(
    [in] DWORD ByteCount,
    [in, size_is(ByteCount)] byte* ByteStream
);

```

ByteCount: Length of *ByteStream* in bytes.

ByteStream: Array of bytes that compose any number of variable-length change notifications structures. Memory for the array is allocated and freed by the caller (that is, the server).

Any variable-length change notification structure in the array starts with a fixed header that contains the fields shown in the following table.

Field name	Data type	Description
size	ULONG	The total size of the structure in bytes.
type	DMNOTIFY_INFO_TYPE	The type of object that changed.
action	LDMACTION	The type of change that the object underwent.
id	LdmObjectId	Identification handle of the object that changed.

Depending on the value of type, the fixed header of the notification structure is followed by one of the following items.

Type	Structure following the fixed header
DMNOTIFY_VOLUME_INFO	VOLUME_INFO
DMNOTIFY_TASK_INFO	TASK_INFO
DMNOTIFY_DL_INFO	DRIVE LETTER_INFO
DMNOTIFY_FS_INFO	FILE_SYSTEM_INFO
DMNOTIFY_SYSTEM_INFO	ULONG
DMNOTIFY_DISK_INFO	If client called Initialize on IVolumeClient interface, then DISK_INFO . If client called Initialize on IVolumeClient3 interface, then DISK_INFO_EX .
DMNOTIFY_REGION_INFO	If client called Initialize on IVolumeClient interface, then REGION_INFO . If client called Initialize on IVolumeClient3 interface, then REGION_INFO_EX .

Note The structures that are transmitted within *ByteStream* are not marshaled in RPC Network Data Representation (NDR) format. They are C structures, and the memory layout and field types are those found on the Windows/Intel 32-bit and 64-bit architectures, and, Windows/AMD 64-bit architecture. These structures are not packed, and padding bytes can exist between successive structure fields to ensure that the field of a given data type begins at a byte offset that is an integer multiple of the type's size with respect to the beginning of the structure. The structures transmitted within *ByteStream* also appear in other interfaces as

RPC-marshaled structures. In these interfaces, the structure fields will be marshaled in NDR format.

The byte stream contains a sequence of one or more notification frames. Each frame is made up of a sequence of the following fields: size, type, action, and a structure of the type specified in the type field. Some of the structures contain character pointer fields. These fields contain pointers to variable-length character strings, and the following technique is used at the server to load the byte stream:

1. The structure is copied one byte at a time from memory into *ByteStream* beginning at first byte after action field. If the structure contains character pointer fields, those fields are omitted.
2. The character strings of the character pointer fields are copied into *ByteStream* following the structure in the order in which they appear in the structure. All strings are null-terminated. There is no padding between the end of the structure and the first string, or between successive strings.

At the client, the following technique is used to parse the byte stream back into the appropriate structures:

1. The notification size, type, and action are parsed from the byte stream.
2. The notification object structure, up through the first string field, is copied out of the byte stream and into the appropriate structure. For the *IVolumeClient* interface, the disk and region structures are **DISK_INFO** and **REGION_INFO**; for the *IVolumeClient3* interface, the structures are **DISK_INFO_EX** and **REGION_INFO_EX**. The client's **ObjectsChanged** implementation MUST switch based on which version of the *IVolumeClient* interface is being used. The client MUST also determine the type of processor architecture for both the server and client. If the architectures are the same, the padding in the client-defined structures will match that used in the server's byte stream. If the architectures are not the same, the client MUST use the proper method for parsing the byte stream, taking into account padding that MAY have been added for alignment purposes on either the client or on the server. For more information, see section 8.

Allocations are done on the client to hold the character strings of the character pointer fields. These fields are copied from *ByteStream* to the client-allocated buffers, and appropriate structure fields are set to point to the client-allocated buffers. All strings are null terminated.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#)); also see section [2.2.1](#) for **HRESULT** values that are predefined by the Disk Management Remote Protocol.

3.1.5 Timer Events

No timers are used by the Disk Management Remote Protocol.

3.1.6 Other Local Events

No other local events require special processing on the client.

3.2 Server Role Details

3.2.1 Abstract Data Model

The following topics contain information that **MUST** be maintained by the server for use in responding to client queries and commands.

3.2.1.1 List of Storage Objects Present in the System

The list contains the following storage objects:

- Disks
 - Hard disks
 - Removable disk units
 - CD-ROM and DVD units
- **Disk regions**
 - Partitions
 - FT volume members
 - Dynamic volume members
 - Free regions
- FT volumes
- Dynamic volumes
- Drive letters
- File system

For each storage object, the list **MUST** contain the following data elements.

id: Unique identifier of type [LdmObjectId](#) associated with the storage object for the entire duration of the server session (defined as one instantiation of the server process from initialization to shutdown). The identifier is assigned by the server and used by the client to refer to the object in the methods of the protocol. The server **MUST NOT** change the identifier and **MUST NOT** assign it to another object until the server shuts down. The identifier does not need to be persisted across server sessions.

type: Type of the storage object (for example, disk, disk region, FT volume, dynamic volume, drive letter, and file system).

LastKnownState: Modification sequence number associated with the storage object. The last known state is used by the protocol to make sure a client has the most up-to-date information about the object before trying to modify the object through one of the protocol commands. The server **MUST** increment the last known state value whenever the object is modified due to a client command or a higher-level event. The server **MUST** also compare the last known state passed by the client with its own last known state before allowing the client to modify the object. If they do not match, the server **MUST** fail the operation.

taskId: Identifier of type **LdmObjectId** of the current task (if any) being executed by the server on the object. This field is not used and can be set to 0.

The list is populated at server initialization time and destroyed at shutdown. Objects are added to, or removed from, the list as a result of client requests or events triggered by the operating system.

3.2.1.2 List of Clients Connected to the Server

For each client connected to the server, the list MUST contain the following data elements.

id: Unique identifier of type [LdmObjectId](#) associated with the client for the entire duration of the client connection. The identifier is assigned by the server and used by the client to identify tasks requested by it when enumerating tasks or receiving notifications related to the progress or failure of tasks. The server MUST NOT change the identifier and MUST NOT assign it to another client until the server shuts down. The identifier does not need to be persisted across separate server sessions.

notifyInterface: Pointer to the [IDMNotify](#) interface implemented by the client to receive notifications from the server. Clients interested in receiving notifications MUST pass such a pointer in the call to `IVolumeClient::Initialize` or `IVolumeClient3::Initialize`. The server MUST retain it for use whenever a notification is to be sent to the client.

The list is empty at server initialization time. Elements are added to, or removed from, the list as a result of clients calling the `Initialize` and `Uninitialize` methods of either [IVolumeClient](#) or [IVolumeClient3](#).

3.2.1.3 List of Tasks Currently Executed on the Server

For each task that is pending on the server, the list MUST contain the following data elements:

taskId: Unique identifier of type [LdmObjectId](#) associated with the task for the entire lifetime of the task. The identifier is assigned by the server and used by the client to map notifications received from the server via `IDMNotify` to specific commands requested by it to follow the progress of those commands. The server MUST NOT change the identifier and MUST NOT assign it to another task until the server shuts down. The identifier does not need to be persisted across separate server sessions.

info: Structure of type [TASK_INFO](#) containing details about the task, such as creation time, the identifier of the client that requested the task, the storage object affected by the task, status, and progress percentage.

The list is empty at server initialization time. Tasks are added to the list as a result of clients requesting configuration operations via the protocol commands. Tasks are removed from the list as they are completed.

3.2.2 Timers

No timers are required.

3.2.3 Initialization

At startup, the server initializes the lists of storage objects, clients, and tasks, as specified in the following topics.

3.2.3.1 List of Storage Objects Present in the System

The server initializes an empty list, and then populates it with all disks, disk regions, FT volumes, dynamic volumes, drive letters, and file systems found in the system. The server MUST assign each object a unique identifier of type [LdmObjectId](#). The **LastKnownState** of each object MUST be initialized with a value at the server's discretion.

3.2.3.2 List of Clients Connected to the Server

The server initializes an empty list.

3.2.3.3 List of Tasks Currently Executed on the Server

The server initializes an empty list.

3.2.4 Message Processing and Sequencing Rules

For all of the following methods, before processing the method, the server SHOULD obtain the identity and authorization information about the client from the underlying DCOM or RPC runtime. All server methods SHOULD impose an authorization policy decision based on the client's identity and authorization information before performing the function.

The method SHOULD fail to complete if there is insufficient authorization. [<15>](#)

All of the parameters to [IVolumeClient](#), [IVolumeClient2](#), [IVolumeClient3](#), [IVolumeClient4](#) and [IDMRemoteServer](#) methods that are not specified as being used MUST be ignored by the server.

3.2.4.1 Higher-Layer Triggered Events

No higher-layer events are processed.

3.2.4.2 Rules for Modifying the List of Storage Objects

A number of protocol message processing steps result in the server modifying its list of storage objects. Possible actions are as follows:

- Add storage object - Done when a new storage object is created as a result of processing the protocol message.
- Delete storage object - Done when a storage object is deleted as a result of processing the protocol message.
- Modify storage object - Done when a storage object is modified as a result of processing the protocol message.

The following subsections list the changes done by the server to the list of storage objects for each of the protocol messages.

When making a change to the list of storage objects, the server MUST follow these rules:

- When adding a storage object, the server MUST generate a unique identifier of type [LdmObjectId](#) for the object, and it MUST initialize the **LastKnownState** field of the object with a value at the server's discretion.
- When updating a storage object, the server MUST increment the **LastKnownState** field of the object.

Any change made to a storage object in the list MUST be accompanied by sending appropriate notification messages to all clients that have registered with the server for receiving notifications via the message [IVolumeClient::Initialize](#) or [IVolumeClient3::Initialize](#).

The following rules MUST be followed with respect to sending notifications.

- When adding a storage object, the server MUST send an IDMNotify::ObjectsChanged notification with action LDMACTION_CREATED for the given storage object.
- When deleting a storage object, the server MUST send an IDMNotify::ObjectsChanged notification with action LDMACTION_DELETE for the given storage object.
- When modifying a storage object, the server MUST send an IDMNotify::ObjectsChanged notification with action LDMACTION_MODIFIED for the given storage object.

Unless otherwise specified in the following subsections, changing the list of storage objects, the manipulation of the **LastKnownState** fields, and sending the notifications to clients MUST all be done by the server before returning the response to the client.

3.2.4.3 Rules for Handling Synchronous and Asynchronous Tasks

A number of protocol messages require the server to execute configuration tasks on the storage objects (for example, delete partition, create volume).

Tasks can be either synchronous or asynchronous. For a synchronous task, the server MUST wait for the task to complete with either success or failure before returning a response to the client. For an asynchronous task, the server MAY return a response to the client before the task completes.

This is the list of methods that MAY be implemented asynchronously. [<16>](#)

- *IVolumeClient::CreatePartitionAssignAndFormat*
- *IVolumeClient3::CreatePartitionAssignAndFormat*
- *IVolumeClient::CreatePartitionAssignAndFormatEx*
- *IVolumeClient3::CreatePartitionAssignAndFormatEx*
- *IVolumeClient::Format*
- *IVolumeClient3::Format*
- *IVolumeClient::CreateVolumeAssignAndFormat*
- *IVolumeClient3::CreateVolumeAssignAndFormat*
- *IVolumeClient::CreateVolumeAssignAndFormatEx*
- *IVolumeClient3::CreateVolumeAssignAndFormatEx*

All the other methods MUST be implemented synchronously.

For the asynchronous methods, the server MUST send periodic notifications to the client after returning the initial response to inform the client about the status and progress of the task. The time interval for these periodic notifications SHOULD be based on two objectives.

- Not flooding the client with unnecessary notifications.

- Providing pertinent information about the ongoing status of the task. [<17>](#)

The protocol messages that require the server to execute configuration tasks receive an output parameter named tinfo of type [TASK_INFO](#).

To process synchronous tasks, the server MUST follow these rules.

1. The server MUST fill the tinfo output parameter as follows:
 - Generate a unique identifier for the task and place it in tinfo.id.
 - Set tinfo.status to the appropriate value of the enumeration REQSTATUS, denoting the success or failure of the task.
 - Set all other fields to 0, unless otherwise specified.
2. The server MUST return the tinfo structure in the response to the client.
3. The server MUST NOT add a new task object to the list of tasks currently running on the server.

For all synchronous method calls that have a **TASK_INFO** structure as an output parameter, the server MUST perform the following extra step (the only methods that do not have to carry out this step are [IVolumeClient::GetEncapsulateDiskInfo](#), [IVolumeClient3::GetEncapsulateDiskInfoEx](#), [IVolumeClient::EnumTasks](#), [IVolumeClient3::EnumTasks](#), [IVolumeClient3::GetTaskDetail](#), and [IVolumeClient::GetTaskDetail](#)):

- Send a task completion notification to the client using the IDMNofity::ObjectsChanged message. The notification MUST be of type DMNOTIFY_TASK_INFO and action LDMACTION_MODIFIED. The status field of the **TASK_INFO** structure MUST be set to the appropriate value of the enumeration REQSTATUS denoting the success or failure of the task.

Note Subsections found under [Protocol Message Details \(section 3.2.4.4\)](#) will explicitly call out any synchronous tasks that require sending task completion notifications.

To process asynchronous tasks, the server MUST follow these rules.

1. The server MUST fill the tinfo output parameter as follows:
 - Generate a unique identifier for the task and place it in tinfo.id.
 - Set tinfo.status to REQ_STARTED.
 - Set the rest of the fields to 0, unless otherwise specified.
2. The server MUST return the tinfo structure in the initial response to the client.
3. The server MUST add a new task object to the list of tasks currently running on the server.
4. Periodically, the server MUST send progress notifications to the clients by using the IDMNofity::ObjectsChanged message. The notifications MUST be of type DMNOTIFY_TASK_INFO and action LDMACTION_MODIFIED. The **percentComplete** field of the **TASK_INFO** structure MUST be set to accurately provide information on the progress of the operation.
5. When the task completes with either success or failure, the server MUST send a final notification to the clients by using the IDMNofity::ObjectsChanged message. The notification MUST be of type DMNOTIFY_TASK_INFO and action LDMACTION_MODIFIED. The **status** field of the **TASK_INFO**

structure MUST be set to the appropriate value of the enumeration REQSTATUS denoting the success or failure of the task.

- When the task is completed, the task object MUST be deleted from the list of tasks that are currently running on the server.

3.2.4.4 Protocol Message Details

3.2.4.4.1 IVolumeClient Methods

This DCOM interface inherits the IUnknown interface. Method opnum field values start with 3; opnum values 0–2 represent the IUnknown_QueryInterface, IUnknown_AddRef, and IUnknown_Release methods, respectively, as specified in [\[MS-DCOM\]](#).

Methods with opnum field values 12, 27, 42, 46, 49, 50 and 59–66 are not invoked across the network, and therefore are not included in this document.

Unless otherwise specified below, all methods MUST return 0 or a nonerror [HRESULT](#) (as specified in [\[MS-ERREF\]](#)) on success, or an implementation-specific nonzero error code on failure (for more information, see section [2.2.1](#) for HRESULT values predefined by the Disk Management Remote Protocol).

Methods in RPC Opnum Order

Method	Description
IVolumeClient::EnumDisks	Opnum: 3
IVolumeClient::EnumDiskRegions	Opnum: 4
IVolumeClient::CreatePartition	Opnum: 5
IVolumeClient::CreatePartitionAssignAndFormat	Opnum: 6
IVolumeClient::CreatePartitionAssignAndFormatEx	Opnum: 7
IVolumeClient::DeletePartition	Opnum: 8
IVolumeClient::WriteSignature	Opnum: 9
IVolumeClient::MarkActivePartition	Opnum: 10
IVolumeClient::Eject	Opnum: 11
Reserved_Opnum12	Opnum: 12
IVolumeClient::FTEnumVolumes	Opnum: 13
IVolumeClient::FTEnumLogicalDiskMembers	Opnum: 14
IVolumeClient::FTDeleteVolume	Opnum: 15
IVolumeClient::FTBreakMirror	Opnum: 16
IVolumeClient::FTResyncMirror	Opnum: 17
IVolumeClient::FTRegenerateParityStripe	Opnum: 18

Method	Description
<u>IVolumeClient::FTReplaceMirrorPartition</u>	Opnum: 19
<u>IVolumeClient::FTReplaceParityStripePartition</u>	Opnum: 20
<u>IVolumeClient::EnumDriveLetters</u>	Opnum: 21
<u>IVolumeClient::AssignDriveLetter</u>	Opnum: 22
<u>IVolumeClient::FreeDriveLetter</u>	Opnum: 23
<u>IVolumeClient::EnumLocalFileSystems</u>	Opnum: 24
<u>IVolumeClient::GetInstalledFileSystems</u>	Opnum: 25
<u>IVolumeClient::Format</u>	Opnum: 26
Reserved27	Opnum: 27
<u>IVolumeClient::EnumVolumes</u>	Opnum: 28
<u>IVolumeClient::EnumVolumeMembers</u>	Opnum: 29
<u>IVolumeClient::CreateVolume</u>	Opnum: 30
<u>IVolumeClient::CreateVolumeAssignAndFormat</u>	Opnum: 31
<u>IVolumeClient::CreateVolumeAssignAndFormatEx</u>	Opnum: 32
<u>IVolumeClient::GetVolumeMountName</u>	Opnum: 33
<u>IVolumeClient::GrowVolume</u>	Opnum: 34
<u>IVolumeClient::DeleteVolume</u>	Opnum: 35
<u>IVolumeClient::AddMirror</u>	Opnum: 36
<u>IVolumeClient::RemoveMirror</u>	Opnum: 37
<u>IVolumeClient::SplitMirror</u>	Opnum: 38
<u>IVolumeClient::InitializeDisk</u>	Opnum: 39
<u>IVolumeClient::UninitializeDisk</u>	Opnum: 40
<u>IVolumeClient::ReConnectDisk</u>	Opnum: 41
Reserved_Opnum42	Opnum: 42
<u>IVolumeClient::ImportDiskGroup</u>	Opnum: 43
<u>IVolumeClient::DiskMergeQuery</u>	Opnum: 44
<u>IVolumeClient::DiskMerge</u>	Opnum: 45
Reserved_Opnum46	Opnum: 46
<u>IVolumeClient::ReAttachDisk</u>	Opnum: 47

Method	Description
<u>IVolumeClient::ReplaceDisk</u>	Opnum: 48
Reserved_Opnum49	Opnum: 49
Reserved_Opnum50	Opnum: 50
<u>IVolumeClient::ReplaceRaid5Column</u>	Opnum: 51
<u>IVolumeClient::RestartVolume</u>	Opnum: 52
<u>IVolumeClient::GetEncapsulateDiskInfo</u>	Opnum: 53
<u>IVolumeClient::EncapsulateDisk</u>	Opnum: 54
<u>IVolumeClient::QueryChangePartitionNumbers</u>	Opnum: 55
<u>IVolumeClient::DeletePartitionNumberInfoFromRegistry</u>	Opnum: 56
<u>IVolumeClient::SetDontShow</u>	Opnum: 57
<u>IVolumeClient::GetDontShow</u>	Opnum: 58
Reserved0	Opnum: 59
Reserved1	Opnum: 60
Reserved2	Opnum: 61
Reserved3	Opnum: 62
Reserved4	Opnum: 63
Reserved5	Opnum: 64
Reserved6	Opnum: 65
Reserved7	Opnum: 66
<u>IVolumeClient::EnumTasks</u>	Opnum: 67
<u>IVolumeClient::GetTaskDetail</u>	Opnum: 68
<u>IVolumeClient::AbortTask</u>	Opnum: 69
<u>IVolumeClient::HrGetErrorData</u>	Opnum: 70
<u>IVolumeClient::Initialize</u>	Opnum: 71
<u>IVolumeClient::Uninitialize</u>	Opnum: 72
<u>IVolumeClient::Refresh</u>	Opnum: 73
<u>IVolumeClient::RescanDisks</u>	Opnum: 74
<u>IVolumeClient::RefreshFileSys</u>	Opnum: 75
<u>IVolumeClient::SecureSystemPartition</u>	Opnum: 76

Method	Description
IVolumeClient::ShutDownSystem	Opnum: 77
IVolumeClient::EnumAccessPath	Opnum: 78
IVolumeClient::EnumAccessPathForVolume	Opnum: 79
IVolumeClient::AddAccessPath	Opnum: 80
IVolumeClient::DeleteAccessPath	Opnum: 81

3.2.4.4.1.1 IVolumeClient::EnumDisks (Opnum 3)

The **EnumDisks** method enumerates the server's **mass storage devices**.

```
HRESULT EnumDisks(
    [out] unsigned long* diskCount,
    [out, size_is(*diskCount)] DISK_INFO** diskList
);
```

diskCount: Number of pointers in *diskList*.

diskList: Pointer to an array of [DISK_INFO](#) structures.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters:

- Verify that *diskCount* and *diskList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all disk objects from the list of storage objects.
2. Allocate a buffer large enough to contain **DISK_INFO** structures that describe all enumerated disks.
3. Populate each **DISK_INFO** structure in the buffer with information about the disk.
4. The buffer MUST be returned to the client in the output parameter *diskList*.
5. The number of **DISK_INFO** structures in the buffer MUST be returned in the output parameter *diskCount*.
6. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.2 IVolumeClient::EnumDiskRegions (Opnum 4)

The **EnumDiskRegions** method enumerates all used and free regions of a specified disk.

```
HRESULT EnumDiskRegions(  
    [in] LdmObjectId diskId,  
    [in, out] unsigned long* numRegions,  
    [out, size_is(*numRegions)] REGION_INFO** regionList  
);
```

diskId: Specifies the object identifier of the disk for which regions are being enumerated.

numRegions: Pointer to the number of regions in *regionList*.

regionList: Pointer to an array of [REGION_INFO](#) structures.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that the disk specified by *diskId* is in the list of storage objects.
- Verify that *numRegions* and *regionList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all disk region objects residing on the specified disk.
2. Allocate a buffer large enough to contain **REGION_INFO** structures describing all regions residing on the disk.
3. The buffer MUST be populated with regions in the ascending order of the byte offset of the region relative to the beginning of the disk.

All fields MUST contain meaningful values. If no volume is associated, **volId** is 0. If there is no associated task, **taskId** is zero.

4. Populate each **REGION_INFO** structure in the buffer with information about the region.
5. The buffer MUST be returned to the client in the output parameter *regionList*.
6. The number of **REGION_INFO** structures in the buffer MUST be returned in the output parameter *numRegions*.
7. Return a response to the client that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.3 IVolumeClient::CreatePartition (Opnum 5)

The **CreatePartition** method creates a partition.

```

HRESULT CreatePartition(
    [in] REGION_SPEC partitionSpec,
    [out] TASK_INFO* tinfo
);

```

partitionSpec: A [REGION_SPEC](#) structure that defines the region type and length to create.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the disk specified by partitionSpec.diskId is in the list of storage objects.
2. Verify that the disk region specified by partitionSpec.regionId is in the list of storage objects, and check if partitionSpec.LastKnownState matches the **LastKnownState** field of the object.
3. Verify that tinfo is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Create a partition on the free disk region specified by partitionSpec.regionId of the disk specified by partitionSpec.diskId.
2. Wait for the partition creation to either succeed or fail.
3. Fill in the *tinfo* output parameter. This is a synchronous task.
 - Field tinfo.storageId MUST be set to the identifier of the disk region object corresponding to the new partition. Other *tinfo* values MUST be set as follows:

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Required if partition is created successfully.
TASK_INFO::createTime	Not required. <18>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <19>

4. Return a response to the client that contains *tinfo* and the status of the operation.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the disk object where the new partition resides to account for the change in region allocation.
2. Create a new disk region object that corresponds to the new partition.
3. Modify or delete the free disk region object where the partition was created to account for the allocation of a new partition in that region.

3.2.4.4.1.4 IVolumeClient::CreatePartitionAssignAndFormat (Opnum 6)

The **CreatePartitionAssignAndFormat** method creates a partition, formats it as a file system, and assigns it a drive letter.

```
HRESULT CreatePartitionAssignAndFormat(  
    [in] REGION_SPEC partitionSpec,  
    [in] wchar_t letter,  
    [in] hyper letterLastKnownState,  
    [in] FILE_SYSTEM_INFO fsSpec,  
    [in] boolean quickFormat,  
    [out] TASK_INFO* tinfo  
);
```

partitionSpec: A [REGION_SPEC](#) structure that defines the type and length of the partition to create.

letter: Drive letter to assign to the new volume, specified as a single, case-insensitive Unicode character.

letterLastKnownState: Drive letter's last known modification sequence number.

fsSpec: A [FILE_SYSTEM_INFO](#) structure that defines the file system to create.

quickFormat: Boolean value that determines if the server will **fully format** or quickly format the file system.

Value	Meaning
FALSE 0	File system will be fully formatted. Full format requires verifying the accessibility of all sectors on the volume.
TRUE 1	File system will be quickly formatted.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server **MUST** validate parameters.

1. Verify that the disk specified by partitionSpec.diskId is in the list of storage objects.
2. Verify that the disk region specified by partitionSpec.regionId is in the list of storage objects, and check if partitionSpec.LastKnownState matches the **LastKnownState** field of the object.
3. Verify that the drive letter object specified by letter is in the list of storage objects, and check if letterLastKnownState matches the **LastKnownState** field of the object.
4. Verify that *tinfo* is not NULL.

If parameter validation fails, the server **MUST** fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server **MUST** process the message as follows:

1. Create a partition on the free disk region specified by partitionSpec.regionId of the disk specified by partitionSpec.diskId.
2. Wait for the partition creation to either succeed or fail.
3. If successful, assign the drive letter specified by letter to the partition.
4. Wait for the drive letter assignment to either succeed or fail.
5. If successful, start formatting the partition with the file system specified by fsSpec, as specified by the *quickFormat* parameter.
6. Fill in the *tinfo* output parameter. This is an asynchronous task.
 - Field tinfo.storageId **MUST** be set to the identifier of the disk region object corresponding to the new partition.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Required.
TASK_INFO::createTime	Not required. <20>
TASK_INFO::clientId	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <21>

7. Return a response to the client containing *tinfo* and the status of the operation.

Note The server **MAY** decide not to wait for the formatting to complete before returning the response to the client. This task is asynchronous and all rules for handling asynchronous tasks apply here.

If the creation of partition is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Modify the disk object where the new partition resides to account for the change in region allocation.
2. Create a new disk region object corresponding to the new partition.
3. Modify or delete the free disk region object where the partition was created to account for the allocation of a new partition in that region.

If the drive letter assignment is successful, the server MUST make the following change to the list of storage objects before returning the response:

- Modify the drive letter object to mark it as in use by the new partition.

If the format operation has been successfully started, the server MUST make the following change to the list of storage objects before returning the response.

- Create a new file system object.

When the formatting is completed, the server MUST make the following change to the list of storage objects.

- Modify the disk region object that corresponds to the partition to account for the change of status.

3.2.4.4.1.5 IVolumeClient::CreatePartitionAssignAndFormatEx (Opnum 7)

The **CreatePartitionAssignAndFormatEx** method creates a partition, formats it as a file system, and assigns it a drive letter and a mount point.

```
HRESULT CreatePartitionAssignAndFormatEx(  
    [in] REGION_SPEC partitionSpec,  
    [in] wchar_t letter,  
    [in] hyper letterLastKnownState,  
    [in] int cchAccessPath,  
    [in, size_is(cchAccessPath)] wchar_t* AccessPath,  
    [in] FILE_SYSTEM_INFO fsSpec,  
    [in] boolean quickFormat,  
    [in] DWORD dwFlags,  
    [out] TASK_INFO* tinfo  
);
```

partitionSpec: A [REGION_SPEC](#) structure that defines the type and length of the partition to create.

letter: Drive letter to assign to the new volume, specified as a single case-insensitive Unicode character.

letterLastKnownState: Drive letter's last known modification sequence number.

cchAccessPath: Length of the *AccessPath* parameter, in characters, including null termination.

AccessPath: Null-terminated Unicode string that specifies the path in which the new file system is being mounted. This parameter is used to supply a mounted folder path, for the case where the new partition will be mounted to a directory on another volume.

fsSpec: A [FILE_SYSTEM_INFO](#) structure that defines the file system to create.

quickFormat: Value that indicates if the server will fully format or quickly format the file system.

Value	Meaning
FALSE 0	File system will be quickly formatted.
TRUE 1	File system will be fully formatted. Full format requires verifying the accessibility of all sectors on the volume.

dwFlags: Bitmap of partition creation flags. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
CREATE_ASSIGN_ACCESS_PATH 0x00000001	Assign the mount point <i>AccessPath</i> to the new partition.

tinfo: Pointer to a [TASK_INFO](#) structure the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

The behavior of the server is identical to that described for [IVolumeClient::CreatePartitionAssignAndFormat](#) with the following difference: Before returning the response to the client, the server MUST create a mount point for the volume under *AccessPath* if instructed by the client to do so.

3.2.4.4.1.6 IVolumeClient::DeletePartition (Opnum 8)

The **DeletePartition** method deletes a specified partition. This is a synchronous task.

```
HRESULT DeletePartition(  
    [in] REGION_SPEC partitionSpec,  
    [in] boolean force,  
    [out] TASK_INFO* tinfo  
);
```

partitionSpec: A [REGION_SPEC](#) structure that specifies the type and length of the partition to delete.

force: Value that determines if deletion of the partition will be forced. If the force parameter is not set, the call will fail if the volume cannot be locked.

Value	Meaning
FALSE 0	Deletion will not be forced if the partition is in use.
TRUE 1	Deletion will be forced.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the disk specified by partitionSpec.diskId is in the list of storage objects.
2. Verify that the disk region specified by partitionSpec.regionId is in the list of storage objects, and check if partitionSpec.LastKnownState matches the **LastKnownState** field of the object.
3. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Attempt to delete the partition specified by partitionSpec.regionId from the disk specified by partitionSpec.diskId, as specified by the *force* parameter.
2. Wait for the partition deletion to either succeed or fail.
3. Fill in the *tinfo* output parameter. This is a synchronous task.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <22>
TASK_INFO::clientId	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <23>

4. Return a response to the client that contains *tinfo* and the status of the operation.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.<24>

1. Modify the disk object where the partition resided to account for the change in region allocation.
2. Delete the disk region object corresponding to the partition.
3. Create a new free region object or modify an adjacent free region object to account for the free space created by the deletion.
4. Modify the drive letter object associated with the partition to mark it as free.
5. Delete the file system object associated with the partition.

3.2.4.4.1.7 IVolumeClient::WriteSignature (Opnum 9)

The **WriteSignature** method writes a disk signature to a specified disk. This is a synchronous task.

```
HRESULT WriteSignature(  
    [in] LdmObjectId diskId,  
    [in] hyper diskLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

diskId: Specifies the object identifier of the target disk for the signature.

diskLastKnownState: Disk's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters:

1. Verify that the disk specified by *diskId* is in the list of storage objects, and check if *diskLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Write an MBRsignature, and initialize the **partition table** of the disk.
2. Wait for the signature writing to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.

TASK_INFO member	Required for this operation
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <25>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <26>

4. Return a response to the client that contains *tinfo* and the status of the operation.

If the operation is successful, the server **MUST** make the following change to the list of storage objects before returning the response.

- Modify the disk object to account for the change of status.

3.2.4.4.1.8 IVolumeClient::MarkActivePartition (Opnum 10)

The **MarkActivePartition** method marks a specified partition as the active partition of the disk. This is a synchronous task.

```
HRESULT MarkActivePartition(
    [in] LdmObjectId regionId,
    [in] hyper regionLastKnownState,
    [out] TASK_INFO* tinfo
);
```

regionId: Specifies the object identifier of the partition to activate.

regionLastKnownState: Partition's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server **MUST** validate parameters.

1. Verify that the disk region specified by *regionId* is in the list of storage objects, and check if *regionLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server **MUST** fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server **MUST** process the message as follows:

1. Mark the partition specified by *regionId* as active in the partition table of its disk.
2. If another partition was marked active on the disk, clear the active flag from it. Only one partition can be active on a given disk at any given time.
3. Wait for the partition activation to either succeed or fail.
4. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. .<27>
TASK_INFO::clientId	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. .<28>

5. Return a response to the client that contains *tinfo* and the status of the operation.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the disk region object that corresponds to the specified partition to account for the change of the active flag.
2. Modify the disk region object that corresponds to the former active partition on the disk to account for the change of the active flag.

3.2.4.4.1.9 IVolumeClient::Eject (Opnum 11)

The **Eject** method ejects a specified removable disk or CD-ROM from the drive enclosure. This is a synchronous task.

```
HRESULT Eject(
    [in] LdmObjectId diskId,
    [in] hyper diskLastKnownState,
    [out] TASK_INFO* tinfo
);
```

diskId: Specifies the object identifier of the media to eject.

diskLastKnownState: Disk's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the disk specified by *diskId* is in the list of storage objects, and check if *diskLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Eject the media from the drive specified by *diskId*.
2. Wait for the eject to succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. .<29>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. .<30>

4. Return a response to the client that contains *tinfo* and the status of the operation.

If the operation is successful, the server makes the following changes to the list of storage objects before returning the response.

1. Modify the disk object to account for the change of status.
2. Delete the disk region object that resides on the disk.
3. Modify the drive letter object that corresponds to the disk region to point to the disk object instead of the disk region object.

3.2.4.4.1.10 IVolumeClient::FTEnumVolumes (Opnum 13)

The **FTEnumVolumes** method enumerates the server's fault-tolerant volumes.

```

HRESULT FTEnumVolumes(
    [in, out] unsigned long* volumeCount,
    [out, size_is(*volumeCount)] VOLUME_INFO** ftVolumeList
);

```

volumeCount: Pointer to the number of elements in *ftVolumeList*.

ftVolumeList: Pointer to an array of [VOLUME_INFO](#) structures. The server allocates this memory and the client frees it.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that *volumeCount* and *ftVolumeList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all FT volume objects from the list of storage objects.
2. Allocate a buffer large enough to contain VOLUME_INFO structures that describe all enumerated FT volumes.
3. Populate each **VOLUME_INFO** structure in the buffer with information about the FT volume.
4. The buffer MUST be returned to the client in the output parameter *ftVolumeList*.
5. The number of **VOLUME_INFO** structures in the buffer MUST be returned in the output parameter *volumeCount*.
6. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.11 IVolumeClient::FTEnumLogicalDiskMembers (Opnum 14)

The **FTEnumLogicalDiskMembers** method enumerates the regions of a specified fault-tolerant volume.

```

HRESULT FTEnumLogicalDiskMembers(
    [in] LdmObjectId volumeId,
    [in, out] unsigned long* memberCount,
    [out, size_is(*memberCount)] LdmObjectId** memberList
);

```

volumeId: Specifies the object identifier of the volume for which regions are being enumerated.

memberCount: Pointer to the number of regions that the volume includes.

memberList: Pointer to an array of [LdmObjectId](#) objects that store member identification handles for the regions in the volume.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that the FT volume specified by *volumeId* is in the list of storage objects.
- Verify that *memberCount* and *memberList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all disk region objects belonging to the FT volume specified by *volumeId* from the list of storage objects.
2. Allocate a buffer large enough to contain all the identifiers of the enumerated disk region objects.
3. Populate the buffer with the identifiers of the disk region objects.
4. The buffer MUST be returned to the client in the output parameter *memberList*.
5. The number of disk region object identifiers in the buffer MUST be returned in the output parameter *memberCount*.
6. Return a response containing the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.12 IVolumeClient::FTDeleteVolume (Opnum 15)

The **FTDeleteVolume** method deletes the fault-tolerant volume specified by *volumeId*. This is a synchronous task.

```
HRESULT FTDeleteVolume(  
    [in] LdmObjectId volumeId,  
    [in] boolean force,  
    [in] hyper volumeLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the volume to delete.

force: Boolean value that indicates if deletion of a partition will be forced. The call to delete will fail if the volume is locked by some other application and this flag is not set.

Value	Meaning
FALSE 0	Deletion will not be forced if the partition is in use.

Value	Meaning
TRUE 1	Deletion of the partition will be forced.

volumeLastKnownState: Volume's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that the FT volume specified by *volumeId* is in the list of storage objects, and check if *volumeLastKnownState* matches the **LastKnownState** field of the object.
- Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Attempt to delete the FT volume specified by *volumeId*, as specified by the *force* parameter. [<31>](#)
2. Wait for the volume deletion to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <32>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <33>

4. Return a response to the client that contains *tinfo* and the status of the operation.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Modify the disk objects where the FT volume resided to account for the change in region allocation.
2. Delete the disk region objects used by the FT volume.
3. Create new free region objects or modify adjacent free region objects to account for the free space created by the deletion.
4. Modify the drive letter object associated with the FT volume to mark it as free.
5. Delete the file system object associated with the FT volume.

3.2.4.4.1.13 IVolumeClient::FTBreakMirror (Opnum 16)

The **FTBreakMirror** method breaks a specified **fault-tolerant mirror set** into two independent partitions. This is a synchronous task.

```
HRESULT FTBreakMirror(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [in] boolean bForce,
    [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the FT mirror set to break.

volumeLastKnownState: Last known modification sequence number of the FT mirror set.

bForce: Boolean value that indicates whether to force removal of the drive letter from the FT mirror set.

Value	Meaning
FALSE 0	The method fails if an error occurs while the drive letter is being removed from the FT mirror set.
TRUE 1	Removal of the drive letter from the FT mirror set is forced.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that the FT volume specified by *volumeId* is in the list of storage objects, and check if *volumeLastKnownState* matches the **LastKnownState** field of the object. Verify that the FT volume is an FT mirror set.
- Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Attempt to remove the drive letter from the FT volume specified by *volumeId*, as specified by the *bForce* parameter.
2. If the removal is successful, or *bForce* is set to TRUE, break the FT volume into two independent partitions. [<34>](#34)
3. If the volume is successfully broken into two partitions, assign the original drive letter of the FT volume to the volume represented by the input *volumeId* parameter.
4. Wait for this sequence of operations to either succeed or fail.
5. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <35>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <36>

6. Return a response to the client containing *tinfo* and the status of the operation.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Modify the disks where the FT volume resided to account for the change of region allocation.
2. Modify the disk region objects used by the FT volume to account for their transformation from members of an FT volume into partitions.
3. Modify the drive letter object associated with the FT volume to account for the migration from the FT volume to one of partitions that results from the break up.
4. Delete the file system object associated with the FT volume.
5. Create the file system objects associated with the partitions that results from the call to break the mirror.

[<37>](#37)

3.2.4.4.1.14 IVolumeClient::FTResyncMirror (Opnum 17)

The **FTResyncMirror** method restores the redundancy of a fault-tolerant mirror set by resynchronizing the members of the mirror. This is a synchronous task.

```
HRESULT FTResyncMirror(  
    [in] LdmObjectId volumeId,  
    [in] hyper volumeLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the FT mirror set being resynchronized.

volumeLastKnownState: Last known modification sequence number of the FT mirror set.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters,

- Verify that the FT volume specified by *volumeId* is in the list of storage objects, and check if *volumeLastKnownState* matches the **LastKnownState** field of the object. Verify that the FT volume is an FT mirror set.
- Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Start the resynchronization of the members of the FT volume specified by *volumeId*.
2. Wait for the resynchronization start to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <38>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.

TASK_INFO member	Required for this operation
TASK_INFO::tflag	Not required. <39>

- Return a response to the client that contains *tinfo* and the status of the operation.

3.2.4.4.1.15 IVolumeClient::FTRegenerateParityStripe (Opnum 18)

The **FTRegenerateParityStripe** method restores the redundancy of a fault-tolerant RAID-5 set by regenerating the parity of the volume. This is a synchronous task.

```
HRESULT FTRegenerateParityStripe(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the FT RAID-5 set for which the parity is being regenerated.

volumeLastKnownState: Last known modification sequence number of the FT RAID-5 set.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that the FT volume specified by *volumeId* is in the list of storage objects, and check if *volumeLastKnownState* matches the **LastKnownState** field of the object. Verify that the FT volume is an FT RAID-5 set.
- Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

- Start the parity regeneration for the FT volume specified by *volumeId*.
- Wait for the parity regeneration start to either succeed or fail.
- Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <40>

TASK_INFO member	Required for this operation
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <41>

4. Return a response to the client that contains *tnfo* and the status of the operation.

3.2.4.4.1.16 IVolumeClient::FTReplaceMirrorPartition (Opnum 19)

The **FTReplaceMirrorPartition** method repairs a fault-tolerant mirror set by replacing the failed member of the set with another partition.

```
HRESULT FTReplaceMirrorPartition(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [in] LdmObjectId oldMemberId,
    [in] hyper oldMemberLastKnownState,
    [in] LdmObjectId newRegionId,
    [in] hyper newRegionLastKnownState,
    [in] DWORD flags,
    [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the FT mirror set to modify.

volumeLastKnownState: FT mirror set's last known modification sequence number.

oldMemberId: This parameter MUST be set to 0 and ignored by the server.

oldMemberLastKnownState: This parameter MUST be set to 0 and ignored by the server.

newRegionId: Specifies the object identifier of the replacement partition.

newRegionLastKnownState: Last known modification sequence number of the replacement partition.

flags: Bitmap of flags for the replacement operation. The value of this field is one of the applicable flags defined below. If 0 is entered, the call will fail if the input *newRegionLastKnownState* is inactive, and will not delete the replacement partition if it fails.

Value	Meaning
FTREPLACE_FORCE 0x00000001	Do not fail the operation if the replacement partition has been changed since <i>newRegionLastKnownState</i> .
FTREPLACE_DELETE_ON_FAIL 0x00000002	Delete the replacement partition if operation fails.

Value	Meaning
0x00000000	Fail the call if the input <i>newRegionLastKnownState</i> is zero, and do not delete the replacement partition if the call fails

tinfo: Pointer to a [TASK_INFO](#) structure that the client uses to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the FT volume specified by *volumeId* is in the list of storage objects, and check if *volumeLastKnownState* matches the **LastKnownState** field of the object. Verify that the FT volume is an FT mirror set.
2. Verify that the partition specified by *newRegionId* is in the list of storage objects, and check if *newRegionLastKnownState* matches the **LastKnownState** field of the object.
 - Ignore *newRegionLastKnownState* if the flag FTREPLACE_FORCE is set in *flags*.
3. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Delete the failed member from the FT volume specified by *volumeId*.
2. Attempt to add the replacement partition specified by *newRegionId* to the FT volume.
3. Wait for the replacement to either succeed or fail.
 - If the replacement failed and the flag FTREPLACE_DELETE_ON_FAIL is set in *flags*, delete the replacement partition.
4. Fill in the *tinfo* output parameter. This is a synchronous task.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <42>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.

TASK_INFO member	Required for this operation
TASK_INFO::tflag	Not required. 43

5. Return a response to the client that contains *tinfo* and the status of the operation.

If the deletion of the failed member is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Modify the FT volume object to account for the change in the list of members.
2. Modify the disk object of the deleted member to account for the change in region allocation.
3. Delete the disk region object that corresponds to the deleted member.

If the addition of replacement partition is successful, the server MUST make the following change to the list of storage objects before returning the response.

- Modify the disk region object that corresponds to the replacement partition to account for transformation from partition to a member of the FT volume.

If the addition of the replacement partition fails and the FTREPLACE_DELETE_ON_FAIL flag is set, the server MUST make the following changes to the list of storage objects before returning the response.

1. Modify the disk object of the deleted replacement partition to account for the change in region allocation.
2. Delete the disk region object that corresponds to the deleted replacement partition.
3. Create a new free region object or modify an adjacent free region object to account for the free space created by the deletion.

3.2.4.4.1.17 IVolumeClient::FTReplaceParityStripePartition (Opnum 20)

The **FTReplaceParityStripePartition** method repairs an FT RAID-5 set by replacing the failed member of the set with another partition.

```
HRESULT FTReplaceParityStripePartition(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [in] LdmObjectId oldMemberId,
    [in] hyper oldMemberLastKnownState,
    [in] LdmObjectId newRegionId,
    [in] hyper newRegionLastKnownState,
    [in] DWORD flags,
    [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the FT RAID-5 set to modify.

volumeLastKnownState: Last known modification sequence number of the FT RAID-5 set.

oldMemberId: This member MUST be set to 0 and ignored by the server.

oldMemberLastKnownState: This member MUST be set to 0 and ignored by the server.

newRegionId: Specifies the object identifier of the replacement partition.

newRegionLastKnownState: Last known modification sequence number of the replacement partition.

flags: Bitmap of flags for the replacement operation. The value of this field is a logical 'OR' of zero or more of the applicable flags defined below.

Value	Meaning
FTREPLACE_FORCE 0x00000001	Do not fail the operation if the replacement partition has been changed since <i>newRegionLastKnownState</i> .
FTREPLACE_DELETE_ON_FAIL 0x00000002	Delete the replacement partition if operation fails.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the FT volume specified by *volumeId* is in the list of storage objects, and check if *volumeLastKnownState* matches the **LastKnownState** field of the object. Verify that the FT volume is an FT RAID-5 set.
2. Verify that the partition specified by *newRegionId* is in the list of storage objects, and check if *newRegionLastKnownState* matches the **LastKnownState** field of the object.
 - Ignore *newRegionLastKnownState* if the flag FTREPLACE_FORCE is set in *flags*.
3. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Delete the failed member from the FT volume specified by *volumeId*.
2. Attempt to add the replacement partition specified by *newRegionId* to the FT volume.
3. Wait for the replacement to either succeed or fail.
 - If the replacement failed and the flag FTREPLACE_DELETE_ON_FAIL is set in *flags*, delete the replacement partition.
4. Fill in the *tinfo* output parameter. This is a synchronous task.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.

TASK_INFO member	Required for this operation
TASK_INFO::createTime	Not required.<44>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required.<45>

5. Return a response to the client that contains *tinfo* and the status of the operation.

If the deletion of failed member is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the FT volume object to account for the change in the list of members.
2. Modify the disk object of the deleted member to account for the change in region allocation.
3. Delete the disk region object that corresponds to the deleted member.

If the addition of replacement partition is successful, the server **MUST** make the following change to the list of storage objects before returning the response.

- Modify the disk region object that corresponds to the replacement partition to account for transformation from partition to a member of the FT volume.

If the addition of replacement partition fails and the FTREPLACE_DELETE_ON_FAIL flag is set, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the disk object of the deleted replacement partition to account for the change in region allocation.
2. Delete the disk region object that corresponds to the deleted replacement partition.
3. Create a new free region object or modify an adjacent free region object to account for the free space created by the deletion.

3.2.4.4.1.18 IVolumeClient::EnumDriveLetters (Opnum 21)

The **EnumDriveLetters** method enumerates the server's drive letters, both used and free. For drive letters that are in use, the method returns the mapping between the drive letter and the volume, partition, or logical drive that uses it.

```
HRESULT EnumDriveLetters(
    [in, out] unsigned long* driveLetterCount,
    [out, size_is(*driveLetterCount)]
    DRIVE_LETTER_INFO** driveLetterList
);
```

driveLetterCount: Pointer to the number of elements returned in *driveLetterList*. This parameter is used only on output.

driveLetterList: Pointer to an array of [DRIVE_LETTER_INFO](#) structures. Memory for the array is allocated by the server and freed by the client.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that *driveLetterCount* and *driveLetterList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Enumerate all drive letter objects from the list of storage objects.
2. Allocate a buffer large enough to contain **DRIVE_LETTER_INFO** structures that describe all enumerated drive letters.
3. Populate each **DRIVE_LETTER_INFO** structure in the buffer with information about the drive letter.
4. The buffer MUST be returned to the client in the output parameter *driveLetterList*.
5. The number of **DRIVE_LETTER_INFO** structures in the buffer MUST be returned in the output parameter *driveLetterCount*.
6. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.19 IVolumeClient::AssignDriveLetter (Opnum 22)

The **AssignDriveLetter** method assigns the specified drive letter to a volume, partition, or logical drive. This is a synchronous task.

```
HRESULT AssignDriveLetter(  
    [in] wchar_t letter,  
    [in] unsigned long forceOption,  
    [in] hyper letterLastKnownState,  
    [in] LdmObjectId storageId,  
    [in] hyper storageLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

letter: Drive letter to assign.

forceOption: Value that indicates if drive letter assignment is forced when it fails.

Value	Meaning
NO_FORCE_OPERATION 0x00000000	If the volume, partition, or logical drive specified by <i>storageId</i> cannot be locked, the operation fails with LDM_E_VOLUME_IN_USE.
FORCE_OPERATION 0x00000001	If the volume, partition, or logical drive specified by <i>storageId</i> cannot be locked, the server will proceed with the operation.

letterLastKnownState: Drive letter's last known modification sequence number.

storageId: Specifies the object identifier of the volume, partition, or logical drive to which the drive letter is being assigned.

storageLastKnownState: Last known modification sequence number of the volume, partition, or logical drive to which the drive letter is being assigned.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the drive letter specified by *letter* is in the list of storage objects, and check if *letterLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that the volume, partition, or logical drive specified by *storageId* is in the list of storage objects; and check if *storageLastKnownState* matches the **LastKnownState** field of the object.
3. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Convert the *letter* parameter to upper case.
2. Attempt to lock the file system (if this is applicable). Locking the file system prevents any other threads from accessing the volume.
3. If the attempt to lock the file system fails, and the NO_FORCE_OPERATION flag was input, the server MUST fail the operation. If the attempt to lock the file system fails, and the FORCE_OPERATION flag was input, ignore the lock failure and continue.
4. Delete any existing drive letter pathname associated with the volume. If the existing drive letter pathname cannot be deleted, the server MUST fail the call.
5. Assign the drive letter *letter* to the storage object.
6. Wait for the drive letter assignment to either succeed or fail.
7. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <46>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <47>

8. Return a response to the client that contains *tinfo* and the status of the operation.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

- Modify the object that corresponds to the old drive letter to mark it as free.
- Modify the object that corresponds to the letter to mark it as associated with the storage object.

3.2.4.4.1.20 IVolumeClient::FreeDriveLetter (Opnum 23)

The **FreeDriveLetter** method unassigns a specified drive letter from a volume, partition, or logical drive on the server. This is a synchronous task.

```
HRESULT FreeDriveLetter(
    [in] wchar_t letter,
    [in] unsigned long forceOption,
    [in] hyper letterLastKnownState,
    [in] LdmObjectId storageId,
    [in] hyper storageLastKnownState,
    [out] TASK_INFO* tinfo
);
```

letter: Drive letter to free.

forceOption: Boolean value that indicates whether to force the freeing of a drive letter.

Value	Meaning
NO_FORCE_OPERATION 0x00000000	If the specified drive letter is assigned to a volume, partition, or logical disk that is in use, contains the paging file, or contains the system directory, the operation fails and returns an error.
FORCE_OPERATION 0x00000001	The specified drive letter is always freed.

letterLastKnownState: Drive letter's last known modification sequence number.

storageId: Specifies the object identifier of the volume, partition, or logical drive to which the letter is assigned.

storageLastKnownState: Last known modification sequence number of the volume, partition, or logical drive to which the drive letter is assigned.

tinfo: Pointer to a [TASK_INFO](#) structure that the client uses to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the drive letter specified by *letter* is in the list of storage objects, and check if *letterLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that the volume, partition, or logical drive specified by *storageId* is in the list of storage objects, and check if *storageLastKnownState* matches the **LastKnownState** field of the object.
3. Verify that the drive letter specified by *letter* is associated with the volume, partition, or logical drive specified by *storageId*.
4. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Attempt to remove the drive letter specified by *letter* from the storage object specified by *storageId*.
2. The behavior of the drive letter removal for volumes, partitions, or logical drives that are in use, contain the paging file, or contain the system directory is controlled by the parameter *forceOption*.[<48>](#)
 - If the parameter is set to NO_FORCE_OPERATION, the removal fails.
 - If the parameter is set to FORCE_OPERATION, the removal succeeds.
3. Wait for the drive letter removal to either succeed or fail.
4. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <49>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.

TASK_INFO member	Required for this operation
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <50>

5. Return response to the client that contains *tinfo* and the status of the operation.

If the operation is successful, the server MUST make the following change to the list of storage objects before returning the response.

- Modify the object that corresponds to the drive letter to mark it as free.

3.2.4.4.1.21 IVolumeClient::EnumLocalFileSystems (Opnum 24)

The **EnumLocalFileSystems** method enumerates the file systems present on the server.

```
HRESULT EnumLocalFileSystems (
    [out] unsigned long* fileSystemCount,
    [out, size_is(*fileSystemCount)]
    FILE_SYSTEM_INFO** fileSystemList
);
```

fileSystemCount: Pointer to the number of elements returned in *fileSystemList*.

fileSystemList: Pointer to an array of **FILE_SYSTEM_INFO** structures that represent the file systems present on the server. Memory for the array is allocated by the server and freed by the client.

Return Values: The method MUST return 0 or a nonerror **HRESULT** on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that *fileSystemCount* and *fileSystemList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all file system objects from the list of storage objects.
2. Allocate a buffer large enough to contain **FILE_SYSTEM_INFO** structures that describe all enumerated file systems.
3. Populate each **FILE_SYSTEM_INFO** structure in the buffer with information about the file system.
4. The buffer MUST be returned to the client in the output parameter *fileSystemList*.

5. The number of **FILE_SYSTEM_INFO** structures in the buffer MUST be returned in the output parameter *fileSystemCount*.
6. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.22 IVolumeClient::GetInstalledFileSystems (Opnum 25)

The **GetInstalledFileSystems** method enumerates the file system types (for example, FAT or NTFS) that the server supports.

```
HRESULT GetInstalledFileSystems(
    [out] unsigned long* fsCount,
    [out, size_is(*fsCount)] IFILE_SYSTEM_INFO** fsList
);
```

fsCount: Pointer to the number of elements returned in *fsList*.

fsList: Pointer to an array of **IFILE_SYSTEM_INFO** structures. Memory for the array is allocated by the server and freed by the client.

Return Values: The method MUST return 0 or a nonerror **HRESULT** on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that *fsCount* and *fsList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all file system types supported by the system.
2. Allocate a buffer large enough to contain **IFILE_SYSTEM_INFO** structures that describe all enumerated file system types.
3. Populate each **IFILE_SYSTEM_INFO** structure in the buffer with information about the file system type.
4. The buffer MUST be returned to the client in the output parameter *fsList*.
5. The number of **IFILE_SYSTEM_INFO** structures in the buffer MUST be returned in the output parameter *fsCount*.
6. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.23 IVolumeClient::Format (Opnum 26)

The **Format** method formats the specified volume, partition, or logical drive with a file system.

```
HRESULT Format(  
    [in] LdmObjectId storageId,  
    [in] FILE_SYSTEM_INFO fsSpec,  
    [in] boolean quickFormat,  
    [in] boolean force,  
    [in] hyper storageLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

storageId: Specifies the object identifier of the volume, partition, or logical drive on which the new file system is being created.

fsSpec: A [FILE_SYSTEM_INFO](#) structure that specifies details about the file system being created.

quickFormat: Boolean value that indicates if the file system will be fully formatted.

Value	Meaning
FALSE 0	File system will be fully formatted. Full format requires verifying the accessibility of all sectors on the volume.
TRUE 1	File system will be quickly formatted.

force: Boolean value that indicates if the file system will be formatted if the volume, partition, or logical drive cannot be locked.

Value	Meaning
FALSE 0	File system will not be formatted unless its underlying storage can be locked.
TRUE 1	File system will be formatted regardless of whether the underlying volume, partition, or logical drive can be locked.

storageLastKnownState: Last known modification sequence number of the volume, partition, or logical drive on which the file system is being created.

tinfo: Pointer to a [TASK_INFO](#) structure that the client uses to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the volume, partition, or logical drive specified by *storageId* is in the list of storage objects; and check if *storageLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Attempt to start formatting the partition with the file system specified by *fsSpec*, as specified by the *quickFormat* parameter and the *force* parameter. [.<51>](#)
2. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. .<52>
TASK_INFO::clientId	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. .<53>

3. Return a response to the client that contains *tinfo* and the status of the operation.

Note The server MAY decide not to wait for the formatting to complete before returning the response to the client. All rules for handling asynchronous tasks apply here.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Modify the storage object specified by *storageId* to account for the change of status.
2. Create a new file system object.

When the formatting is completed, the server MUST make the following change to the list of storage objects.

- Modify the storage object specified by *storageId* to account for the change of status.

3.2.4.4.1.24 IVolumeClient::EnumVolumes (Opnum 28)

The **EnumVolumes** method enumerates the dynamic volumes of the server.

```
HRESULT EnumVolumes(  
    [in, out] unsigned long* volumeCount,  
    [out, size_is(*volumeCount)] VOLUME_INFO** LdmVolumeList  
);
```

volumeCount: Pointer to the number of elements returned in LdmVolumeList.

LdmVolumeList: Pointer to an array of [VOLUME_INFO](#) structures representing the dynamic volumes of the server. Memory for the array is allocated by the server and freed by the client.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters:

- Verify that *volumeCount* and *LdmVolumeList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all dynamic volume objects from the list of storage objects.
2. Allocate a buffer large enough to contain **VOLUME_INFO** structures that describe all enumerated dynamic volumes.
3. Populate each **VOLUME_INFO** structure in the buffer with information about the dynamic volume.
4. The buffer MUST be returned to the client in the output parameter *LdmVolumeList*.
5. The number of **VOLUME_INFO** structures in the buffer MUST be returned in the output parameter *volumeCount*.
6. Return a response containing the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.25 IVolumeClient::EnumVolumeMembers (Opnum 29)

The **EnumVolumeMembers** method enumerates the regions of the specified dynamic volume.

```
HRESULT EnumVolumeMembers(  
    [in] LdmObjectId volumeId,  
    [in, out] unsigned long* memberCount,  
    [out, size_is(*memberCount)] LdmObjectId** memberList  
);
```

volumeId: Specifies the object identifier of the volume for which regions are being enumerated.

memberCount: Pointer to the number of disk regions returned in *memberList*.

memberList: Array of [LdmObjectId](#) objects that store the identification handles of the regions. Memory for the array is allocated by the server and freed by the client.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that the dynamic volume specified by *volumeId* is in the list of storage objects.
- Verify that *memberCount* and *memberList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all disk region objects belonging to the dynamic volume from the list of storage objects.
2. Allocate a buffer large enough to contain the identifiers of all enumerated disk region objects.
3. Populate the buffer with the identifiers of all enumerated disk region objects.
4. The buffer MUST be returned to the client in the output parameter *memberList*.
5. The number of identifiers in the buffer MUST be returned in the output parameter *memberCount*.
6. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.26 IVolumeClient::CreateVolume (Opnum 30)

The **CreateVolume** method creates a dynamic volume on the specified list of disks. This is a synchronous task.

```
HRESULT CreateVolume(
    [in] VOLUME_SPEC volumeSpec,
    [in] unsigned long diskCount,
    [in, size_is(diskCount)] DISK_SPEC* diskList,
    [out] TASK_INFO* tinfo
);
```

volumeSpec: A [VOLUME_SPEC](#) structure that defines the parameters of the volume to create.

diskCount: Number of elements passed in *diskList*.

diskList: Array of [DISK_SPEC](#) structures that specifies the disks to be used by the volume.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that *diskCount* is not 0 and *diskList* is not NULL.
- For each **DISK_SPEC** structure in *diskList*, verify that the disk specified by *diskId* is in the list of storage objects; and check if *lastKnownState* matches the **LastKnownState** field of the object.
- Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Create the dynamic volume as follows:

- The layout, length, and number of members of the volume are determined by the fields layout and length of parameter *volumeSpec* and parameter *diskCount*.
- The members of the volume are created on the disks passed in *diskList*.
- The approximate length of each member is determined by the field length of the corresponding **DISK_SPEC** structure passed in *diskList*.
- If the field **needContiguous** is set to TRUE in a **DISK_SPEC** structure passed in *diskList*, the server MUST allocate a contiguous disk region for the corresponding member. Otherwise, the server MAY allocate several non-contiguous disk regions.

2. Wait for the volume creation to either succeed or fail.

3. Fill in the *tinfo* output parameter.

- Field **tinfo.storageId** MUST be set to the identifier of the new dynamic volume object.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Required.
TASK_INFO::createTime	Not required. <54>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <55>

4. Return a response to the client containing *tinfo* and the status of the operation.

5. Send the task completion notification.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Create a new dynamic volume object.
2. Modify the disk objects where the new volume resides to account for the change in region allocation.
3. Create new disk region objects that correspond to the volume members.

4. Modify or delete the free disk region objects where the volume members were created to account for the allocation of volume members in those regions.

3.2.4.4.1.27 IVolumeClient::CreateVolumeAssignAndFormat (Opnum 31)

The **CreateVolumeAssignAndFormat** method creates a dynamic volume on the specified list of disks, assigns a drive letter to it, and formats it with a file system.

```
HRESULT CreateVolumeAssignAndFormat(  
    [in] VOLUME_SPEC volumeSpec,  
    [in] unsigned long diskCount,  
    [in, size_is(diskCount)] DISK_SPEC* diskList,  
    [in] wchar_t letter,  
    [in] hyper letterLastKnownState,  
    [in] FILE_SYSTEM_INFO fsSpec,  
    [in] boolean quickFormat,  
    [out] TASK_INFO* tinfo  
);
```

volumeSpec: A [VOLUME_SPEC](#) structure that defines the volume to create.

diskCount: Number of elements passed in *diskList*.

diskList: Array of [DISK_SPEC](#) structures that specifies the disks to be used by the volume. Memory for the array is allocated and freed by the client.

letter: Drive letter to assign to the new volume. If no drive letter is needed for the volume, the value of this field MUST be a two-byte NULL character or the Unicode SPACE character.

letterLastKnownState: Drive letter's last known modification sequence number.

fsSpec: A [FILE_SYSTEM_INFO](#) structure that defines the file system to create.

quickFormat: Boolean value that indicates whether the server will fully format or quickly format the file system.

Value	Meaning
FALSE 0	File system will be fully formatted. Full format requires verifying the accessibility of all sectors on the volume.
TRUE 1	File system will be quickly formatted.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that *diskCount* is not 0 and *diskList* is not NULL.
2. For each **DISK_SPEC** structure in *diskList*, verify that the disk specified by *diskId* is in the list of storage objects; and check if *lastKnownState* matches the **LastKnownState** field of the object.

3. Verify that the drive letter object, if specified by letter, is in the list of storage objects, and check if letterLastKnownState matches the **LastKnownState** field of the object.
4. Verify that *tinfo* is not NULL.

If parameter validation fails, the server **MUST** fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server **MUST** process the message as follows:

1. Attempt to create the dynamic volume as follows:
 - The layout, length, and number of members of the volume are determined by the field's layout, length, and memberCount of parameter *volumeSpec*.
 - The members of the volume **MUST** be created on the disks passed in *diskList*.
 - The length of each member is determined by the field length of the corresponding **DISK_SPEC** structure passed in *diskList*.
 - If the field **needContiguous** is set to TRUE in a **DISK_SPEC** structure passed in *diskList*, the server **MUST** allocate a contiguous disk region for the corresponding member. Otherwise, the server **MAY** allocate several non-contiguous disk regions.
2. Wait for the volume creation to either succeed or fail.
3. If successful, assign the drive letter, if specified by letter, to the volume.
4. Wait for the drive letter assignment to either succeed or fail.
5. If successful, start formatting the volume with the file system specified by *fsSpec*, as specified by the *quickFormat* parameter.
6. Fill in the *tinfo* output parameter.
 - Field **tinfo.storageId** **MUST** be set to the identifier of the dynamic volume object.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Required.
TASK_INFO::createTime	Not required. <56>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <57>

7. Return a response to the client that contains *tinfo* and the status of the operation.

8. Send the task completion notification.

Note The server MAY decide not to wait for the formatting to complete before returning the response to the client. All rules for handling asynchronous tasks apply here.

If the volume creation is successful, the server MUST make the following changes to the list of storage objects before returning the response:

1. Create a new dynamic volume object.
2. Modify the disk objects where the new volume resides to account for the change in region allocation.
3. Create new disk region objects that correspond to the volume members.
4. Modify or delete the free disk region objects where the volume members were created to account for the allocation of volume members in those regions.

If the drive letter assignment is successful, the server MUST make the following change to the list of storage objects before returning the response.

- Modify the drive letter object to mark it as in-use by the new volume.

If the format operation is successfully started, the server MUST make the following change to the list of storage objects before returning the response.

- Create a new file system object.

When the formatting is completed, the server MUST make the following change to the list of storage objects.

- Modify the dynamic volume object to account for the change of status.

3.2.4.4.1.28 **IVolumeClient::CreateVolumeAssignAndFormatEx (Opnum 32)**

The **CreateVolumeAssignAndFormatEx** method creates a dynamic volume on the specified list of disks, assigns a drive letter and/or a mount point to it, and formats it with a file system.

```
HRESULT CreateVolumeAssignAndFormatEx(  
    [in] VOLUME_SPEC volumeSpec,  
    [in] unsigned long diskCount,  
    [in, size_is(diskCount)] DISK_SPEC* diskList,  
    [in] wchar_t letter,  
    [in] hyper letterLastKnownState,  
    [in] int cchAccessPath,  
    [in, size_is(cchAccessPath)] wchar_t* AccessPath,  
    [in] FILE_SYSTEM_INFO fsSpec,  
    [in] boolean quickFormat,  
    [in] DWORD dwFlags,  
    [out] TASK_INFO* tinfo  
);
```

volumeSpec: A [VOLUME_SPEC](#) structure that defines the volume to create.

diskCount: Number of elements passed in *diskList*.

diskList: Array of [DISK_SPEC](#) structures that specifies the disk to be used by the volume.

letter: Drive letter to assign to the new volume. If no drive letter is needed for the volume, the value of this field **MUST** be a two-byte NULL character or the Unicode SPACE character.

letterLastKnownState: Drive letter's last known modification sequence number.

cchAccessPath: Length of *AccessPath*, including null termination.

AccessPath: Null-terminated path in which the new file system is being mounted. The server **MUST** ignore this parameter if CREATE_ASSIGN_ACCESS_PATH is not set in *dwFlags*.

fsSpec: A [FILE_SYSTEM_INFO](#) structure that defines the file system to create. All fields **MUST** be filled out unless otherwise specified in section [3.1.4.1.3](#).

quickFormat: Value that indicates whether the file system will be fully formatted or quickly formatted.

Value	Meaning
FALSE 0	File system will be fully formatted. Full format requires verifying the accessibility of all sectors on the volume.
TRUE 1	File system will be quickly formatted.

dwFlags: Bitmap of volume creation flags. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
CREATE_ASSIGN_ACCESS_PATH 0x00000001	Assign the mount point <i>AccessPath</i> to the new volume. If the flag is not set, the parameter <i>AccessPath</i> is ignored.

info: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

The behavior of the server is almost identical to the one described for [IVolumeClient::CreateVolumeAssignAndFormat](#). The only difference is that if the client specifies the CREATE_ASSIGN_ACCESS_PATH flag after attempting to assign the drive letter, the server **MUST** attempt to create a mount point for the volume under *AccessPath*, and wait for the mount point assignment to succeed or fail

3.2.4.4.1.29 IVolumeClient::GetVolumeMountName (Opnum 33)

The **GetVolumeMountName** method retrieves the mount name for a volume, partition, or logical drive.

```
HRESULT GetVolumeMountName (
    [in] LdmObjectId volumeId,
    [out] unsigned long* cchMountName,
    [out, size_is(*cchMountName)]
    WCHAR** mountName
);
```


volumeId: Specifies the object identifier of the volume for which the mount name is being retrieved.

cchMountName: Pointer to the length of *mountName*, including null termination.

mountName: Pointer to the null-terminated mount name of the volume, in Unicode characters, in the format `\\?\Volume{guid}` (note that the question mark is literal, not a wildcard). Memory for the string is allocated by the server and freed by the client.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the volume, partition, or logical drive specified by *volumeId* is in the list of storage objects.
2. Verify that *cchMountName* and *mountName* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Retrieve the mount name of the volume, partition, or logical drive specified by *volumeId* in the format `\\?\Volume{guid}` (note that the question mark is literal, not a wildcard).
2. Allocate a buffer large enough to contain the mount name, including null termination.
3. Populate the buffer with the mount name, including null-termination.
4. The buffer MUST be returned to the client in the output parameter *mountName*.
5. The number of characters in the buffer, including null-termination, MUST be returned in the output parameter *cchMountName*.
6. Return response containing the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.30 **IVolumeClient::GrowVolume (Opnum 34)**

The **GrowVolume** method increases the length of a specified dynamic volume by appending extents from the specified disks to it. This is a synchronous task.

```
HRESULT GrowVolume(  
    [in] LdmObjectId volumeId,  
    [in] VOLUME_SPEC volumeSpec,  
    [in] unsigned long diskCount,  
    [in, size_is(diskCount)] DISK_SPEC* diskList,  
    [in] boolean force,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the volume whose size is being changed.

volumeSpec: A [VOLUME_SPEC](#) structure that defines the parameters of the changed volume, including its new expected length.

diskCount: Number of elements passed in *diskList*.

diskList: Array of [DISK_SPEC](#) structures that specifies the list of **disk extents** to be appended to the volume. Memory for the array is allocated and freed by the client. All fields **MUST** be filled out.

force: Boolean value that determines whether the volume is extended or not in case it cannot be locked.

Value	Meaning
FALSE 0	Volume is not extended unless it is locked.
TRUE 1	Volume is extended whether it is locked or unlocked.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server **MUST** validate parameters.

1. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects, and check if the field **volumeSpec.lastKnownState** matches the field **LastKnownState** of the object.
2. Verify that *diskCount* is not 0 and *diskList* is not NULL.
3. For each **DISK_SPEC** structure in *diskList*, verify that the disk specified by *diskId* is in the list of storage objects; and check if **lastKnownState** matches the **LastKnownState** field of the object.
4. Verify that *tinfo* is not NULL.

If parameter validation fails, the server **MUST** fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server **MUST** process the message as follows:

1. Attempt to grow the dynamic volume as follows:
 - The new length and number of members of the volume are determined by the field's length and memberCount of parameter *volumeSpec*.
 - New members of the volume **MUST** be created on the disks passed in *diskList* and concatenated to the volume, as specified by the *force* parameter.
 - The length of each member is determined by the field length of the corresponding **DISK_SPEC** structure passed in *diskList*.
2. Wait for the volume growth to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required.<58>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required.<59>

4. Return a response to the client containing *tinfo* and the status of the operation.

5. Send the task completion notification.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the dynamic volume object to account for the change in size and number of members.
2. Modify the disk objects where the new volume members were created to account for the change in region allocation.
3. Create new disk region objects that corresponds to the new volume members.
4. Modify or delete the free disk region objects where the new volume members were created to account for the allocation of volume members in those regions.<60>

3.2.4.4.1.31 IVolumeClient::DeleteVolume (Opnum 35)

The **DeleteVolume** method deletes the specified dynamic volume. This is a synchronous task.

```
HRESULT DeleteVolume(
    [in] LdmObjectId volumeId,
    [in] boolean force,
    [in] hyper volumeLastKnownState,
    [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the volume to delete.

force: Value that indicates if deletion of the volume will be forced if the volume is in use.

Value	Meaning
FALSE 0	Deletion will not be forced if the volume is in use.

Value	Meaning
TRUE 1	Deletion will be forced.

volumeLastKnownState: Volume's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects, and check if **volumeLastKnownState** matches the **LastKnownState** field of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Attempt to delete the dynamic volume specified by *volumeId*, as specified by the *force* parameter.
2. Wait for the volume deletion to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <61>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <62>

4. Return a response to the client that contains *tinfo* and the status of the operation.
5. Send the task completion notification.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Delete the dynamic volume object.
2. Modify the disks where the volume resided to account for the change in region allocation.
3. Delete the disk region objects that correspond to the volume.
4. Create new free region objects, or modify adjacent free region objects, to account for the free space created by the deletion.
5. Modify the drive letter object associated with the volume to mark it as free.
6. Delete the file system object associated with the volume. [<63>](#63)

3.2.4.4.1.32 IVolumeClient::AddMirror (Opnum 36)

The **AddMirror** method adds a mirror to the specified dynamic volume. This is a synchronous task.

```
HRESULT AddMirror(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [in] DISK_SPEC diskSpec,
    [in, out] int* diskNumber,
    [out] int* partitionNumber,
    [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the volume to which the mirror is being added.

volumeLastKnownState: Volume's last known modification sequence number.

diskSpec: A [DISK_SPEC](#) structure that defines the disk to add as mirror.

diskNumber: Unused. This parameter MUST be set to 0 by the client and MUST be ignored by the server.

partitionNumber: Pointer to the partition number of the newly added mirror.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects, and check if the field *volumeLastKnownState* matches the field **LastKnownState** of the object.
2. Verify that the disk specified by **diskSpec.diskId** is in the list of storage objects, and check if *diskSpec.lastKnownState* matches the **LastKnownState** field of the object.
3. Verify that *partitionNumber* is not NULL.
4. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Add a mirror to the dynamic volume as follows:
 - A new member of the volume MUST be created on the disk specified by **diskSpec.diskId** and added to the volume as mirror.
 - The length of the member is determined by the field **diskSpec.length**.
 - If the field **diskSpec.needContiguous** is set to TRUE, the server MUST allocate a contiguous disk region for the new member. Otherwise, the server MAY allocate several non-contiguous disk regions.
2. Wait for the mirror addition to either succeed or fail.
3. Fill the *partitionNumber* output parameter as follows:
 - If the dynamic volume is a boot volume, set *partitionNumber* to the partition number of the new volume members.
 - Otherwise, set *partitionNumber* to 0.
4. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <64>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <65>

5. Return a response to the client containing *tinfo*, *partitionNumber*, and the status of the operation.
6. Send the task completion notification.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Modify the dynamic volume object to account for the change in layout and number of members.
2. Modify the disk object where the new volume member was created to account for the change in region allocation.
3. Create new disk region objects that correspond to the new volume member.

4. Modify or delete the free disk region objects where the new volume member was created to account for the allocation of the volume member in those regions.

3.2.4.4.1.33 IVolumeClient::RemoveMirror (Opnum 37)

The **RemoveMirror** method removes a mirror from a dynamic volume. This is a synchronous task.

```
HRESULT RemoveMirror(  
    [in] LdmObjectId volumeId,  
    [in] hyper volumeLastKnownState,  
    [in] LdmObjectId diskId,  
    [in] hyper diskLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the mirrored volume from which the disk is being removed.

volumeLastKnownState: Volume's last known modification sequence number.

diskId: Specifies the object identifier of the disk being removed from the volume.

diskLastKnownState: Last known modification sequence number of the disk being removed from the volume.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects, and check if the field **volumeLastKnownState** matches the field **LastKnownState** of the object. Verify that the dynamic volume is a mirrored one.
2. Verify that the disk specified by *diskId* is in the list of storage objects, and check if *diskLastKnownState* matches the **LastKnownState** field of the object. Verify that the disk specified by *diskId* is in the mirror volume specified by *volumeId*.
3. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Delete the mirror of the volume specified by *volumeId* residing on the disk specified by *diskId*.
2. Wait for the mirror removal to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required.<66>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required.<67>

4. Return response to the client containing *tinfo* and the status of the operation.
5. Send the task completion notification.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the dynamic volume object to account for the change in layout and number of members.
2. Modify the disks where the deleted volume member resided to account for the change in region allocation.
3. Delete the disk region objects that correspond to the deleted volume member.
4. Create new free region objects or modify adjacent free region objects to account for the free space created by the deletion.

3.2.4.4.1.34 IVolumeClient::SplitMirror (Opnum 38)

The **SplitMirror** method splits a dynamic mirrored volume into two independent **simple volumes**, one with the identifier and drive letter of the original volume and the other with a different identifier and drive letter. This is a synchronous task.

```
HRESULT SplitMirror(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [in] LdmObjectId diskId,
    [in] hyper diskLastKnownState,
    [in] wchar_t letter,
    [in] hyper letterLastKnownState,
    [in, out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the volume to split.

volumeLastKnownState: Volume's last known modification sequence number.

diskId: Specifies the object identifier of the disk to break away from the volume specified by *volumeId*.

diskLastKnownState: Last known modification sequence number of the disk to split off.

letter: Drive letter to assign to the disk identified by *diskId*. If no drive letter is needed for the volume, the value of this field MUST be either a two-byte NULL character or the Unicode SPACE character.

letterLastKnownState: Last known modification sequence number of the drive letter that is being assigned to the disk to split.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects, and check if the field **volumeLastKnownState** matches the field **LastKnownState** of the object.
2. Verify that the disk specified by *diskId* is in the list of storage objects and check if *diskLastKnownState* matches the **LastKnownState** field of the object.
3. Verify that the drive letter, if specified by *letter*, is in the list of storage objects and check if *letterLastKnownState* matches the **LastKnownState** field of the object.
4. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows.

1. Attempt to remove the mirror of the volume specified by *volumeId* residing on the disk specified by *diskId*. If the client sets the `TASK_INFO::error` parameter to `LDM_DEVICE_IN_USE`, the server MUST remove the mirror of the volume, even if the volume is in use. If the client does not set the `TASK_INFO::error` parameter to `LDM_DEVICE_IN_USE` and if the volume is in use, the server MUST fail the operation immediately, returning the `LDM_E_VOLUME_IN_USE` error as its response to the client.
2. Transform the removed member into a standalone dynamic volume.
3. If successful, assign the drive letter, if specified by *letter*, to the new volume.
4. Wait for the above sequence to either succeed or fail.
5. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
<code>TASK_INFO::id</code>	Required.
<code>TASK_INFO::storageId</code>	Not required.
<code>TASK_INFO::createTime</code>	Not required. <68>

TASK_INFO member	Required for this operation
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <69>

6. Return a response to the client containing *tinfo* and the status of the operation.

7. Send the task completion notification.

If the removal of mirror is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the old dynamic volume object to account for the change in layout and number of members.
2. Create a new dynamic volume object for the new volume.
3. Modify the disk region objects split from the old volume to account for their migration to the new volume.

If the drive letter assignment is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

- Modify the drive letter object, if specified by letter, to mark it as in use by the new volume.
- Create a new file system object for the new volume. [<70>](#)

3.2.4.4.1.35 IVolumeClient::InitializeDisk (Opnum 39)

The **InitializeDisk** method converts an uninitialized disk into a dynamic disk. This is a synchronous task.

```
HRESULT InitializeDisk(
    [in] LdmObjectId diskId,
    [in] hyper diskLastKnownState,
    [out] TASK_INFO* tinfo
);
```

diskId: Specifies the object identifier of the disk to initialize.

diskLastKnownState: Disk's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the disk specified by *diskId* is in the list of storage objects and check if *diskLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Initialize the disk specified by *diskId* with an empty MBRpartition table and write an MBRsignature to it.
2. If successful, convert the disk to a dynamic disk.
3. Wait for the conversion to either succeed or fail.
4. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required.<71>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required.<72>

5. Return a response to the client containing *tinfo* and the status of the operation.
6. Send the task completion notification.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Modify the disk object to account for the change in type.
2. Delete disk region objects residing on the uninitialized disk.
3. Create disk region objects residing on the dynamic disk.

3.2.4.4.1.36 IVolumeClient::UninitializeDisk (Opnum 40)

The **UninitializeDisk** method converts an empty dynamic disk back to a basic disk. This is a synchronous task.

```

HRESULT UninitializeDisk(
    [in] LdmObjectId diskId,
    [in] hyper diskLastKnownState,
    [out] TASK_INFO* tinfo
);

```

diskId: Specifies the object identifier of the disk to uninitialize.

diskLastKnownState: Disk's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the disk specified by *diskId* is in the list of storage objects and check if *diskLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that the disk specified by *diskId* is empty.
3. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Convert the dynamic disk specified by *diskId* to a basic disk.
2. Wait for the conversion to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <73>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <74>

4. Return a response to the client that contains *tinfo* and the status of the operation.

5. Send the task completion notification.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the disk object to account for the change in type.
2. Delete disk region objects that reside on the dynamic disk.
3. Create disk region objects that reside on the basic disk.

3.2.4.4.1.37 IVolumeClient::ReConnectDisk (Opnum 41)

The **ReConnectDisk** method reactivates a failed dynamic disk, bringing the disk and the volumes residing on it **online**. This is a synchronous task.

```
HRESULT ReConnectDisk(  
    [in] LdmObjectId diskId,  
    [out] TASK_INFO* tinfo  
);
```

diskId: Specifies the object identifier of the disk to reactivate.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server **MUST** validate parameters:

1. Verify that the dynamic disk specified by *diskId* is in the list of storage objects.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server **MUST** fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server **MUST** process the message as follows:

1. Bring the failed dynamic disk specified by *diskId* online.
 - Bring any dynamic volumes that reside on the dynamic disk online if possible.
 - Start resynchronization for any mirrored and RAID-5 volumes that reside on the disk.
2. Wait for the operation to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.

TASK_INFO member	Required for this operation
TASK_INFO::createTime	Not required.<75>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required.<76>

4. Return a response to the client that contains *tinfo* and the status of the operation.
5. Send the task completion notification.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Modify the disk object to account for the change in status.
2. Modify the volume objects that reside on the disk to account for the change in status.
3. Modify drive letter objects to mark them as in use by the volumes brought online.
4. Create file system objects for the volumes brought online.

3.2.4.4.1.38 IVolumeClient::ImportDiskGroup (Opnum 43)

The **ImportDiskGroup** method imports a foreign dynamic disk group as the primary disk group of the server. This is a synchronous task.

```
HRESULT ImportDiskGroup(
    [in] int cchDgid,
    [in, size_is(cchDgid)] byte* dgid,
    [out] TASK_INFO* tinfo
);
```

cchDgid: Size of *dgid* in characters, including null-termination.

dgid: Null-terminated string that contains the universally unique identifier (UUID) of the disk group to import. This parameter is generated by converting a GUID to a null-terminated ASCII string, and then treating the resulting string as a byte array.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that *dgid* is a valid disk group id that belongs to a foreign dynamic disk group.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Make the foreigndisk group specified by *dgid* the primary disk group of the system.
 - Bring all dynamic disks and volumes that belong to the disk group online.
2. Wait for the operation to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <77>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <78>

4. Return a response to the client that contains *tinfo* and the status of the operation.
5. Send the task completion notification.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

1. Modify the disk objects of the disk group to account for the change in status.
2. Create new dynamic volume objects that correspond to the imported volumes.
3. Create new disk region objects that correspond to the imported volumes.
4. Modify drive letter objects to mark them as in use by the imported volumes (if the volumes have drive letters).
5. Create file system objects for the imported volumes (if the volumes are formatted with file systems).

3.2.4.4.1.39 IVolumeClient::DiskMergeQuery (Opnum 44)

The **DiskMergeQuery** method gathers disk and volume information needed to merge a foreign dynamic disk group into the primary disk group of the server. This is a synchronous task.

```
HRESULT DiskMergeQuery(  
    [in] int cchDgid,  
    [in, size_is(cchDgid)] byte* dgid,  
    [in] int numDisks,  
    [in, size_is(numDisks)] LdmObjectId* diskList,  
    [out] hyper* merge_config_tid,  
    [out] int* numRids,  
    [out, size_is(*numRids)] hyper** merge_dm_rids,  
    [out] int* numObjects,  
    [out, size_is(*numObjects)] MERGE_OBJECT_INFO** mergeObjectInfo,  
    [in, out] unsigned long* flags,  
    [out] TASK_INFO* tinfo  
);
```

cchDgid: Size of *dgid* in characters, including null-termination.

dgid: Null-terminated string containing the universally unique identifier (UUID) of the disk group to be merged. This parameter is generated by converting a GUID to a null-terminated ASCII string and then treating the resulting string as a byte array.

numDisks: Number of disks passed in *diskList*.

diskList: Array of object identifiers of type [LdmObjectId](#) that specify the disks of the *dgid* group to be merged.

merge_config_tid: Pointer to the modification sequence number of the disk group to be merged.

numRids: Pointer to the number of elements returned in *merge_dm_rids*.

merge_dm_rids: Pointer to an array of disk records representing the disks that will be merged.

numObjects: Number of elements returned in *mergeObjectInfo*.

mergeObjectInfo: Pointer to an array of [MERGE_OBJECT_INFO](#) structures that contain information about the volumes that will be merged.

flags: Disk merge query flags. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
DSKMERGE_IN_NO_UNRELATED 0x00000001	Do not retrieve merge information for volumes of the foreign disk group that do not have extents on <i>diskList</i> . This is an input-only flag.
DSKMERGE_OUT_NO_PRIMARY_DG 0x00000001	The machine does not have a primary disk group. This is an output-only flag.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that *dgid* is a valid disk group id belonging to a foreign dynamic disk group.
2. Verify that the disk objects specified by *diskList* are in the list of storage objects and belong to the disk group specified by *dgid*.
3. Verify that *merge_config_tid* is not NULL.
4. Verify that *numRids* and *merge_dm_rids* are not NULL.
5. Verify that *numObjects* and *mergeObjectInfo* are not NULL.
6. Verify that *flags* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. The modification sequence number of the disk group specified by *dgid* MUST be returned in the output parameter *merge_config_tid*.
2. Enumerate all dynamic disk records from the configuration of the disk group.
3. Allocate a buffer large enough to contain the identifiers of the enumerated dynamic disk records.
4. Populate the buffer with the identifiers of the enumerated dynamic disk records.
5. The buffer MUST be returned to the client in the output parameter *merge_dm_rids*.
6. The number of identifiers in the buffer MUST be returned in the output parameter *numRids*.
7. If DSKMERGE_IN_NO_UNRELATED flag is set, attempt to enumerate only the volumes that have extent on *diskList*. If DSKMERGE_IN_NO_UNRELATED flag is not set, enumerate all volumes belonging to the disk group that will be merged.
8. Allocate a second buffer large enough to contain **MERGE_OBJECT_INFO** structures that describe the enumerated volumes.
9. Populate each **MERGE_OBJECT_INFO** structure in the second buffer with information about the volume.
10. The second buffer MUST be returned to the client in the output parameter *mergeObjectInfo*.
11. The number of **MERGE_OBJECT_INFO** structures in the second buffer MUST be returned to the client in the output parameter *numObjects*.
12. If the machine does not have a primary disk group, the server MUST set the DSKMERGE_OUT_NO_PRIMARY_DG flag in the output parameter *flags*.
13. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message. [<79>](#)

3.2.4.4.1.40 IVolumeClient::DiskMerge (Opnum 45)

The **DiskMerge** method merges a foreign disk group into the primary disk group of the server. This is a synchronous task.

```
HRESULT DiskMerge(  
    [in] int cchDgid,  
    [in, size_is(cchDgid)] byte* dgid,  
    [in] int numDisks,  
    [in, size_is(numDisks)] LdmObjectId* diskList,  
    [in] hyper merge_config_tid,  
    [in] int numRids,  
    [in, size_is(numRids)] hyper* merge_dm_rids,  
    [out] TASK_INFO* tinfo  
);
```

cchDgid: Size of *dgid* in characters, including null-termination.

dgid: Null-terminated string that contains the universally unique identifier (UUID) of the disk group to be merged.

numDisks: Number of disks passed in *diskList*.

diskList: Array of object identifiers of type [LdmObjectId](#) that specify the disks to be merged from the *dgid* group.

merge_config_tid: Last known modification sequence number of the disk group to be merged.

numRids: Number of elements passed in *merge_dm_rids*.

merge_dm_rids: Array of disk records for the disks in *diskList*. Memory for the array is allocated and freed by the client.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message the server MUST validate parameters:

1. Verify that *dgid* is a valid disk group id that belongs to a foreign dynamic disk group.
2. Verify that the disk objects specified by *diskList* are in the list of storage objects and belong to the disk group specified by *dgid*.
3. Verify that *merge_config_tid* matches the modification sequence number of the disk group specified by *dgid*.
4. Verify that the disk records specified in *merge_dm_rids* exist in the configuration of the disk group specified by *dgid*.
5. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Merge the foreigndisk group specified by *dgid* into the primary disk group of the system.
 - Bring all dynamic disks and volumes belonging to the foreign disk group online.
2. Wait for the merge to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <80>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <81>

4. Return a response to the client containing *tinfo* and the status of the operation.
5. Send the task completion notification.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response:

1. Modify the disk objects of the foreigndisk group to account for the change in status.
2. Create new dynamic volume objects that correspond to the imported volumes.
3. Create new disk region objects that correspond to the imported volumes.
4. Modify drive letter objects to mark them as in use by the imported volumes (if the volumes have drive letters).
5. Create file system objects for the imported volumes (if the volumes are formatted with file systems).

3.2.4.4.1.41 IVolumeClient::ReAttachDisk (Opnum 47)

The **ReAttachDisk** method reattaches the specified dynamic disk, bringing the volumes of the disk back online after reconnecting the disk device to the server. This is a synchronous task.

```
HRESULT ReAttachDisk(  
    [in] LdmObjectId diskId,  
    [in] hyper diskLastKnownState,  
    [out] TASK_INFO* tinfo
```

);

diskId: Specifies the object identifier of the disk to reattach.

diskLastKnownState: Disk's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that the dynamic disk specified by *diskId* is in the list of storage objects, and check if *diskLastKnownState* matches the **LastKnownState** field of the object.
- Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Bring the dynamic disk specified by *diskId* online.
 - Mark the disk as being present.
 - Bring any dynamic volumes that reside on the disk online.
2. Wait for the operation to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <82>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <83>

4. Return a response to the client containing *tinfo* and the status of the operation.

5. Send the task completion notification.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the disk object to account for the change in status.
2. Modify the volume objects that reside on the disk to account for the change in status.
3. Modify drive letter objects that correspond to the volumes brought online to mark them as free.
4. Create file system objects for the volumes brought online (if the volumes are formatted with file systems).

3.2.4.4.1.42 **IVolumeClient::ReplaceDisk (Opnum 48)**

The **ReplaceDisk** method replaces a specified dynamic disk with another disk in the primary disk group of the server. This is a synchronous task. [<84>](#84)

```
HRESULT ReplaceDisk(  
    [in] LdmObjectId oldDiskId,  
    [in] hyper oldDiskLastKnownState,  
    [in] LdmObjectId newDiskId,  
    [in] hyper newDiskLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

oldDiskId: Specifies the object identifier of the disk to replace.

oldDiskLastKnownState: Last known modification sequence number of the disk to replace.

newDiskId: Specifies the object identifier of the replacement disk.

newDiskLastKnownState: Last known modification sequence number of the replacement disk.

tinfo: Pointer to a [TASK_INFO](#TASK_INFO) structure that the client can use to track the request's progress.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#HRESULT) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#MS-ERREF); see also section [2.2.1](#2.2.1) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server **MUST** validate parameters.

1. Verify that the dynamic disk specified by *oldDiskId* is in the list of storage objects, and check if *oldDiskLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that the dynamic disk specified by *newDiskId* is in the list of storage objects, and check if *newDiskLastKnownState* matches the **LastKnownState** field of the object.
3. Verify that *tinfo* is not NULL.

If parameter validation fails, the server **MUST** fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server **MUST** process the message as follows:

1. Replace the dynamic disk specified by *oldDiskId* with the dynamic disk specified by *newDiskId*:

1. Recreate the volume members of the old dynamic disk on the new dynamic disk.
2. Delete the volume members that reside on the old dynamic disk.
3. Start synchronization for any mirrored and RAID-5 volumes repaired in the process.
2. Wait for the disk replacement to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. .<85>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. .<86>

4. Return a response to the client containing *tinfo* and the status of the operation.
5. Send the task completion notification.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify both old and new disk objects to account for the change in region allocation.
2. Modify the volume objects repaired by the operation to account for the change in status.

3.2.4.4.1.43 IVolumeClient::ReplaceRaid5Column (Opnum 51)

The **ReplaceRaid5Column** method repairs a dynamic RAID-5 volume by replacing the failed member of the volume with a specified disk. This is a synchronous task.

```
HRESULT ReplaceRaid5Column(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [in] LdmObjectId newDiskId,
    [in] hyper diskLastKnownState,
    [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the volume in which to replace the member.

volumeLastKnownState: Last known modification sequence number of the RAID-5 volume.

newDiskId: Specifies the object identifier of the replacement disk.

diskLastKnownState: Replacement disk's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects, and check if *volumeLastKnownState* matches the field **LastKnownState** of the object. Verify that the volume is RAID-5.
2. Verify that the disk specified by *newDiskId* is in the list of storage objects, and check if *diskLastKnownState* matches the **LastKnownState** field of the object.
3. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Replace the failed member of the RAID-5 volume specified by *volumeId*:
 1. Remove and delete the failed member of the volume.
 2. Create a new member of the volume on the disk specified by *newDiskId*.
 3. Start a task to regenerate the parity of the volume.
2. Wait for the member replacement to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <87>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <88>

4. Return a response to the client containing *tinfo* and the status of the operation.

5. Send the task completion notification.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the dynamic volume object to account for the change in status and list of members.
2. Modify the disk object where the new volume member was created to account for the change in region allocation.
3. Create new disk region objects that corresponds to the new volume member.
4. Modify or delete the free disk region objects where the new volume member was created to account for the allocation of the volume member in those regions.
5. Modify the disk object that corresponds to the deleted member to account for the change in region allocation.
6. Delete disk region objects that corresponds to the deleted member.
7. Create new free region objects or modify adjacent free region objects to account for the free space created by the deletion of the old member.

3.2.4.4.1.44 **IVolumeClient::RestartVolume (Opnum 52)**

The **RestartVolume** method attempts to bring a dynamic volume back online. This is a synchronous task.

```
HRESULT RestartVolume(  
    [in] LdmObjectId volumeId,  
    [in] hyper volumeLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the volume to restart.

volumeLastKnownState: Volume's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server **MUST** validate parameters.

1. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects, and check if *volumeLastKnownState* matches the field **LastKnownState** of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server **MUST** fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server **MUST** process the message as follows:

1. Bring the volume specified by *volumeId* online.

2. Wait for the operation to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <89>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <90>

4. Return a response to the client that contains *tinfo* and the status of the operation.
5. Send the task completion notification.

If the operation is successful the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the dynamic volume object to account for the change in status.
2. Modify the drive letter object to mark it as in use by the volume (if the volume has a drive letter).
3. Create a file system object for the volume (if the volume is formatted with a file system).

3.2.4.4.1.45 IVolumeClient::GetEncapsulateDiskInfo (Opnum 53)

The **GetEncapsulateDiskInfo** method gathers the information needed to convert the specified basic disks to dynamic disks. This is a synchronous task.

```

HRESULT GetEncapsulateDiskInfo(
    [in] unsigned long diskCount,
    [in, size_is(diskCount)] DISK_SPEC* diskSpecList,
    [out] unsigned long* encapInfoFlags,
    [out] unsigned long* affectedDiskCount,
    [out, size_is(*affectedDiskCount)]
        DISK_INFO** affectedDiskList,
    [out, size_is(*affectedDiskCount)]
        unsigned long** affectedDiskFlags,
    [out] unsigned long* affectedVolumeCount,
    [out, size_is(*affectedVolumeCount)]
        VOLUME_INFO** affectedVolumeList,
    [out] unsigned long* affectedRegionCount,
    [out, size_is(*affectedRegionCount)]
        REGION_INFO** affectedRegionList,

```

```
[out] TASK_INFO* tinfo
);
```

diskCount: Number of elements passed in the *diskSpecList* array.

diskSpecList: Array of [DISK_SPEC](#) structures that specify the disks to be encapsulated.

encapInfoFlags: Bitmap of flags that returns information about encapsulating the disks specified in *diskSpecList*. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
ENCAP_INFO_CANT_PROCEED 0x00000001	Encapsulation for disk will not succeed. The other flags specify the reason.
ENCAP_INFO_NO_FREE_SPACE 0x00000002	Volume manager could not find sufficient free space on the disk for encapsulation.
ENCAP_INFO_BAD_ACTIVE 0x00000004	Disk contains an active partition from which the current operating system was started.
ENCAP_INFO_UNKNOWN_PART 0x00000008	Volume manager was unable to determine the type of a partition on the disk.
ENCAP_INFO_FT_UNHEALTHY 0x00000010	Disk contains an FT set volume that is not functioning properly.
ENCAP_INFO_FT_QUERY_FAILED 0x00000020	Volume manager was unable to obtain information about an FT set volume on the disk.
ENCAP_INFO_FT_HAS_RAID5 0x00000040	Disk is part of an FT RAID-5 set, which this interface does not support for encapsulation.
ENCAP_INFO_FT_ON_BOOT 0x00000080	Disk is both part of an FT set volume and bootable, which this interface does not support for encapsulation.
ENCAP_INFO_REBOOT_REQD 0x00000100	Encapsulation of the disk requires a restart of the computer.
ENCAP_INFO_CONTAINS_FT 0x00000200	Disk is part of an FT set volume.
ENCAP_INFO_VOLUME_BUSY 0x00000400	Disk is currently in use.
ENCAP_INFO_PART_NR_CHANGE 0x00000800	Encapsulation of the disk requires modification of the boot configuration.

affectedDiskCount: Pointer to the number of disks that will be affected by the encapsulation.

affectedDiskList: Pointer to an array of new [DISK_INFO](#) structures that represent the disks that will be affected by the encapsulation. Memory for the array is allocated by the server and freed by the client.

affectedDiskFlags: Pointer to an array of bitmaps of flags that provide information about the disks that will be affected by the encapsulation. Memory for the array is allocated by the

server and freed by the client. The value of this field is a logical 'OR' of 0 or more of the applicable flags defined below.

Value	Meaning
CONTAINS_FT 0x00000001	Disk contains an FT set volume.
CONTAINS_RAID5 0x00000002	Disk contains part of an FT RAID-5 set.
CONTAINS_REDISTRIBUTION 0x00000004	Disk contains an unknown volume type.
CONTAINS_BOOTABLE_PARTITION 0x00000008	Disk contains a bootable partition.
CONTAINS_LOCKED_PARTITION 0x00000010	Disk contains a locked partition .
CONTAINS_NO_FREE_SPACE 0x00000020	Disk is full.
CONTAINS_EXTENDED_PARTITION 0x00000040	Disk contains an empty partition.
PARTITION_NUMBER_CHANGE 0x00000080	A partition number on the disk has changed.
CONTAINS_BOOTINDICATOR 0x00000100	Disk contains the active partition.
CONTAINS_BOOTLOADER 0x00000200	Disk contains the boot loader .
CONTAINS_SYSTEMDIR 0x00000400	Partition contains the system directory.
CONTAINS_MIXED_PARTITIONS 0x00000800	Partition contains different types of partitions.

affectedVolumeCount: Pointer to the number of volumes that will be affected by the encapsulation.

affectedVolumeList: Pointer to an array of [VOLUME_INFO](#) structures that represent the volumes that will be affected by the encapsulation. Memory for the array is allocated by the server and freed by the client.

affectedRegionCount: Pointer to the number of regions that will be affected by the encapsulation.

affectedRegionList: Pointer to an array of [REGION_INFO](#) structures that represent the regions that will be affected by the encapsulation. Memory for the array is allocated by the server and freed by the client.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that *diskCount* is not 0 and *diskSpecList* is not NULL.
2. For each **DISK_SPEC** structure specified in *diskSpecList*, verify that the disk specified by *diskId* is in the list of storage objects; and check if *lastKnownState* matches the **LastKnownState** field of the object.
3. Verify that *encapInfoFlags* is not NULL.
4. Verify that *affectedDiskCount*, *affectedDiskList*, and *affectedDiskFlags* are not NULL.
5. Verify that *affectedVolumeCount* and *affectedVolumeList* are not NULL.
6. Verify that *affectedRegionCount* and *affectedRegionList* are not NULL.
7. Verify that *flags* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Identify other basic disks from the list of storage objects that need to be encapsulated together with the basic disks specified by *diskSpecList*.
2. Allocate a buffer large enough to contain **DISK_INFO** structures that describe all basic disks that need to be encapsulated together (including the disks specified by *diskSpecList*).
3. Populate each **DISK_INFO** structure in the buffer with information about the disk.
4. The buffer MUST be returned to the client in the output parameter *affectedDiskList*.
5. The number of **DISK_INFO** structures in the buffer MUST be returned to the client in the output parameter *affectedDiskCount*.
6. Allocate a second buffer large enough to contain bitmaps of flags, one for each disk returned in *affectedDiskList*, that describe disk conditions that are of interest to clients in the context of encapsulation.
7. Populate the second buffer with the bitmaps of flags of the disks.
8. The second buffer MUST be returned to the client in the output parameter *affectedDiskList*. Note that the list of elements in the buffer is the same as the value returned in *affectedDiskCount*.
9. Enumerate all the FT volumes that reside on the disks returned in *affectedDiskList* from the list of storage objects.
10. Allocate a third buffer large enough to contain **VOLUME_INFO** structures that describe the enumerated FT volumes.
11. Populate each **VOLUME_INFO** structure in the third buffer with information about the FT volume.
12. The third buffer MUST be returned to the client in the output parameter *affectedVolumeList*.

13. The number of **VOLUME_INFO** structures in the third buffer MUST be returned to the client in the output parameter *affectedVolumeCount*.
14. Enumerate all the disk regions that reside on the disks returned in *affectedDiskList* from the list of storage objects.
15. Allocate a fourth buffer large enough to contain **REGION_INFO** structures that describe the enumerated disk regions.
16. Populate each **REGION_INFO** structure in the fourth buffer with information about the disk region.
17. The fourth buffer MUST be returned to the client in the output parameter *affectedRegionList*.
18. The number of **REGION_INFO** structures in the fourth buffer MUST be returned to the client in the output parameter *affectedRegionCount*.
19. Populate a 32-bit-signed integer bitmap of flags describing conditions that will prevent the overall encapsulation to proceed, or might be of interest to the client in the context of encapsulation. If the encapsulation cannot proceed, the server MUST set the **ENCAP_INFO_CANT_PROCEED** flag, and then set other flags as appropriate to account for the reasons why the encapsulation is not possible.
20. The bitmap of flags MUST be returned to the client in the output parameter *encapInfoFlags*.
21. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.46 IVolumeClient::EncapsulateDisk (Opnum 54)

The **EncapsulateDisk** method converts the specified basic disks to dynamic disks. This is a synchronous task.

```
HRESULT EncapsulateDisk(
    [in] unsigned long affectedDiskCount,
    [in, size_is(affectedDiskCount)]
        DISK_INFO* affectedDiskList,
    [in] unsigned long affectedVolumeCount,
    [in, size_is(affectedVolumeCount)]
        VOLUME_INFO* affectedVolumeList,
    [in] unsigned long affectedRegionCount,
    [in, size_is(affectedRegionCount)]
        REGION_INFO* affectedRegionList,
    [out] unsigned long* encapInfoFlags,
    [out] TASK_INFO* tinfo
);
```

affectedDiskCount: The number of elements passed in the *affectedDiskList* array.

affectedDiskList: An array of [DISK_INFO](#) structures that specify the disks to be encapsulated.

affectedVolumeCount: The number of elements passed in the *affectedVolumeList* array.

affectedVolumeList: An array of [VOLUME_INFO](#) structures that represent the volumes affected by the encapsulation.

affectedRegionCount: The number of elements passed in the *affectedRegionList* array.

affectedRegionList: An array of [REGION_INFO](#) structures that represent the regions affected by the encapsulation.

encapInfoFlags: Bitmap of flags that provide information about the encapsulation. The value of this field is a logical 'OR' of zero or more of the applicable flags defined below.

Value	Meaning
ENCAP_INFO_CANT_PROCEED 0x00000001	Encapsulation for disk did not succeed. The other flags specify the reason.
ENCAP_INFO_NO_FREE_SPACE 0x00000002	The volume manager could not find sufficient free space on the disk for encapsulation.
ENCAP_INFO_BAD_ACTIVE 0x00000004	The disk contains an active partition from which the current operating system was not started.
ENCAP_INFO_UNKNOWN_PART 0x00000008	The volume manager was unable to determine the type of a partition on the disk.
ENCAP_INFO_FT_UNHEALTHY 0x00000010	The disk contains an unhealthy FT set volume.
ENCAP_INFO_FT_QUERY_FAILED 0x00000020	The volume manager was unable to obtain information about an FT set volume on the disk.
ENCAP_INFO_FT_HAS_RAID5 0x00000040	The disk is part of an FT RAID-5 set, which this interface does not support for encapsulation.
ENCAP_INFO_FT_ON_BOOT 0x00000080	The disk is part of an FT set volume and bootable, which this interface does not support for encapsulation.
ENCAP_INFO_REBOOT_REQD 0x00000100	Encapsulation of the disk requires a restart of the computer.
ENCAP_INFO_CONTAINS_FT 0x00000200	The disk is part of an FT set volume.
ENCAP_INFO_VOLUME_BUSY 0x00000400	The disk is currently in use.
ENCAP_INFO_PART_NR_CHANGE 0x00000800	Encapsulation of the disk requires modification of the boot configuration.

tfinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. For each **DISK_INFO** structure specified by *affectedDiskList*, verify that the disk specified by *diskId* is in the list of storage objects and *lastKnownState* matches the **LastKnownState** field of the object.
2. For each **VOLUME_INFO** structure specified by *affectedVolumeList*, verify that the basic disk volume specified by *id* is in the list of storage objects and *lastKnownState* matches the **LastKnownState** field of the object.
3. Verify that the list of basic disk volumes specified by *affectedVolumeList* matches the set of basic disk volumes that reside on the disks specified by *affectedDiskList*.
4. For each **REGION_INFO** structure specified by *affectedRegionList*, verify that the disk region specified by *regionId* is in the list of storage objects and *lastKnownState* matches the **LastKnownState** field of the object.
5. Verify that the list of disk regions specified by *affectedRegionList* matches the set of disk regions that reside on the disks specified by *affectedDiskList*.
6. Verify that *encapInfoFlags* is not NULL.
7. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Convert the basic disks specified by *affectedDiskList* to dynamic.
 - All partitions and logical drives that reside on the basic disk are converted to dynamic volumes.
2. Wait for the conversion to either succeed or fail.
3. Fill in the *encapInfoFlags* output parameter.
4. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <91>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <92>

5. Return a response to the client containing *tinfo* and the status of the operation.
6. Send the task completion notification.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the converted disk objects to account for the change in type.
2. Create new dynamic volume objects that correspond to the new dynamic volumes.
3. Create new disk region objects for the new dynamic disks.
4. Delete disk region objects of the old basic disks.
5. Modify drive letter objects to account for the change of volume owning them.
6. Modify file system objects to account for the change of volume owning them.

If the boot partition is among the disks being encapsulated, the server **MUST** store boot partition change information on persistent storage (registry).

The information **MUST** contain the old (pre-encapsulation) and new (post-encapsulation) partition number of the boot partition. The information is useful in case the client sends an

[IVolumeClient::QueryChangePartitionNumbers](#) message.

IVolumeClient::QueryChangePartitionNumbers will return the original partition number and the new partition number. This information **MAY** be used to update boot settings if necessary. [<93>](#)

3.2.4.4.1.47 IVolumeClient::QueryChangePartitionNumbers (Opnum 55)

The **QueryChangePartitionNumbers** method retrieves information about the partition number change that results when a boot partition is encapsulated.

```
HRESULT QueryChangePartitionNumbers(  
    [out] int* oldPartitionNumber,  
    [out] int* newPartitionNumber  
);
```

oldPartitionNumber: Pointer to the partition number of the boot volume before the encapsulation operation.

newPartitionNumber: Pointer to the partition number of the boot volume after the encapsulation operation.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server **MUST** validate parameters.

- Verify that *oldPartitionNumber* and *newPartitionNumber* are not NULL.

If parameter validation fails, the server **MUST** fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server **MUST** compose a response to the client as follows:

1. Retrieve the boot partition change information from persistent storage (registry).
2. The old (pre-encapsulation) partition number of the boot partition MUST be returned in the output parameter *oldPartitionNumber*. Return 0 if there is no boot partition change information to report.
3. The new (post-encapsulation) partition number of the boot partition MUST be returned in the output parameter *newPartitionNumber*. Return 0 if there is no boot partition change information to report.
4. Return a response containing the output parameters mentioned above and the status of the operation.

3.2.4.4.1.48 IVolumeClient::DeletePartitionNumberInfoFromRegistry (Opnum 56)

The **DeletePartitionNumberInfoFromRegistry** method deletes the boot partition number change history from persistent storage.

```
HRESULT DeletePartitionNumberInfoFromRegistry();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

The server MUST process the message as follows:

1. Delete the boot partition change information from persistent storage (registry).
2. Wait for the deletion to succeed or fail.
3. Return a response to the client that contains the status of the operation.

3.2.4.4.1.49 IVolumeClient::SetDontShow (Opnum 57)

See Windows behavior note. [<94>](#)

```
HRESULT SetDontShow(
    [in] boolean bSetNoShow
);
```

bSetNoShow: Boolean value that determines whether the New Disk Wizard is enabled or disabled.

Value	Meaning
FALSE 0	Enable new Disk Wizard. This value is the default. It indicates that the user has not selected the check box in the New Disk Wizard to request that the wizard not to be displayed in the future.
TRUE 1	Disable new Disk Wizard. This value indicates that the user has selected the check box in the New Disk Wizard to request that the wizard not to be displayed in the future.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

The server MUST process the message as follows:

1. Save the *bSetNoShow* setting on persistent storage (registry). The value of the setting MUST be returned to the client in subsequent calls to [IVolumeClient::GetDontShow](#).
2. Wait for the operation to succeed or fail.
3. Return a response to the client that contains the status of the operation.

3.2.4.4.1.50 IVolumeClient::GetDontShow (Opnum 58)

See Windows behavior note. [<95>](#)

```
HRESULT GetDontShow(  
    [out] boolean* bGetNoShow  
);
```

bGetNoShow: Boolean value that indicates whether the new Disk Wizard is enabled or disabled.

Value	Meaning
FALSE 0	New Disk Wizard is enabled. This value is the default. It indicates that the user has not selected the check box in the New Disk Wizard to request that the wizard not to be displayed in the future.
TRUE 1	New Disk Wizard is disabled. This value indicates that the user has selected the check box in the New Disk Wizard to request that the wizard not to be displayed in the future.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that *bGetNoShow* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Retrieve the *bGetNoShow* setting saved on persistent storage (registry) during the most recent call to [IVolumeClient::SetDontShow](#).
2. The setting MUST be returned to the client in the output parameter *bGetNoShow*.
3. Return a response that contains the output parameters mentioned above and the status of the operation.

3.2.4.4.1.51 IVolumeClient::EnumTasks (Opnum 67)

The **EnumTasks** method enumerates the tasks currently running on the server.

```

HRESULT EnumTasks(
    [in, out] unsigned long* taskCount,
    [out, size_is(*taskCount)] TASK_INFO** taskList
);

```

taskCount: Number of elements returned in the *taskList* array. The client SHOULD set the value of this parameter as zero.

taskList: Array of [TASK_INFO](#) structures that describe the tasks running on the server. Memory for the array is allocated by the server and freed by the client.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that *taskCount* and *taskList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all task objects from the list of tasks currently running on the server.
2. Allocate a buffer large enough to contain **TASK_INFO** structures that describe all enumerated tasks.
3. Populate each **TASK_INFO** structure in the buffer with information about the task.
4. The buffer MUST be returned to the client in the output parameter *taskList*.
5. The number of **TASK_INFO** structures in the buffer MUST be returned in the output parameter *taskCount*.
6. Return a response containing the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of tasks currently running on the server as part of processing this message.

3.2.4.4.1.52 IVolumeClient::GetTaskDetail (Opnum 68)

The **GetTaskDetail** method retrieves information about a task running on the server.

```

HRESULT GetTaskDetail(
    [in] LdmObjectId id,
    [in, out] TASK_INFO* tinfo
);

```

id: Specifies the object identifier of the task for which to retrieve information.

tinfo: A [TASK_INFO](#) structure that describes the operation currently being performed by *id*. The client SHOULD set values of all members of **TASK_INFO** structure as zero.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the task specified by *id* is in the list of tasks currently running on the server.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Fill a **TASK_INFO** structure with the status of the task.
2. The filled **TASK_INFO** structure MUST be returned in the output parameter *tinfo*.
3. Return response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of tasks currently running on the server as part of processing this message.

3.2.4.4.1.53 IVolumeClient::AbortTask (Opnum 69)

The **AbortTask** method aborts a task running on the server.

```
HRESULT AbortTask(  
    [in] LdmObjectId id  
);
```

id: Specifies the object identifier of the task to be aborted.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that the task specified by *id* is in the list of tasks currently running on the server.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Cancel the task specified by *id*. Server MUST attempt to stop all activity related to the task.
2. Wait for the cancellation to succeed or fail.
3. Return a response to the client containing the status of the operation.
4. If successful, send task completion notification and delete the task object from the list of tasks currently running on the server.

3.2.4.4.1.54 IVolumeClient::HrGetErrorData (Opnum 70)

The **HrGetErrorData** method retrieves user-readable error information associated with an **HRESULT** error code. [<96>](#)

```
HRESULT HrGetErrorData(  
    [in] HRESULT hr,  
    [in] DWORD dwFlags,  
    [out] DWORD* pdwStoredFlags,  
    [out] int* pcszw,  
    [out, string, size_is(*pcszw,)]  
    wchar_t*** prgszw  
);
```

hr: The **HRESULT** error code from which error information is retrieved.

dwFlags: Bitmap of retrieval flags. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
ERRFLAG_NOREMOVE 0x00020000	Do not delete the error information.
ERRFLAG_IGNORETAG 0x00040000	Retrieve the error information even if it was not produced for this client.

pdwStoredFlags: Pointer to a bitmap of error flags. There are no flags defined. [<97>](#)

pcszw: Pointer to the number of strings returned in *prgszw*.

prgszw: Pointer to an array of strings that contain error information for the **HRESULT**. For example, for error LDM_E_CRASHDUMP_PAGEFILE_BOOT_SYSTEM_VOLUME, the string would be: "The request cannot be completed because the volume is open or in use. It MAY be configured as a system, boot, or pagefile volume, or, to hold a crashdump file." Memory for the array is allocated by the server and freed by the client.

Return Values: The method MUST return 0 or a nonerror **HRESULT** on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that *pdwStoredFlags* is not NULL.
2. Verify that *pcszw* and *prgszw* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate user-readable error messages that can be used to better explain the error code specified by *hr* to the end user.

- Such error messages are produced by the server during failed client requests. Since the server is not allowed to return error messages in the response to the protocol message that failed, it MAY save such information in a store in preparation for returning it to the client in a subsequent call to **IVolumeClient::HrGetErrorData**.
 - **Note** Servers are not mandated to support producing error messages. Servers that choose to support producing error messages are free to decide which protocol messages produce error messages and which do not. [<98>](#)
2. Unless the flag `ERRFLAG_IGNORETAG` is set in the input parameter *dwFlags*, the server MUST filter out error messages produced as a result of a failed requests initiated by other clients.
 3. For each one of the enumerated error messages, allocate a buffer large enough to contain the entire error message, including null-termination. Populate each buffer with the corresponding error message.
 4. Allocate an array large enough to contain pointers to each one of the error message buffers.
 5. Populate the array with pointers to the error message buffers.
 6. The array MUST be returned to the client in the output parameter *prgszw*.
 7. The number of pointers to error messages in the array MUST be returned to the client in the output parameter *pcszw*.
 8. Output parameter *pdwStoredFlags* MUST be set to 0.
 9. Unless the flag `ERRFLAG_NOREMOVE` is set in the input parameter *dwFlags*, the server MUST delete the error messages for the error code specified by *hr* from its store.
 10. Return a response that contains the output parameters mentioned above and the status of the operation.

3.2.4.4.1.55 IVolumeClient::Initialize (Opnum 71)

The **Initialize** method initializes the dialogue between the client and the server.

```
HRESULT Initialize(
    [in] IUnknown* notificationInterface,
    [out] unsigned long* ulIDLVersion,
    [out] DWORD* pdwFlags,
    [out] LdmObjectId* clientId,
    [in] unsigned long cRemote
);
```

notificationInterface: Pointer to the client's IUnknown interface from which the server can query the [IDMNotify](#) interface used for sending notifications to the client.

ulIDLVersion: The value of `LDM_IDL_VERSION` found in the IDL file with which the server was built. [<99>](#)

pdwFlags: Bitmap of information flags about the server. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
SYSFLAG_SERVER 0x00000001	Server is running on Windows Server 2003 and Windows 2000 Server.
SYSFLAG_ALPHA 0x00000002	Server is running on an Alpha processor. <100>
SYSFLAG_SYSPART_SECURE 0x00000004	System partition for the server is secure. <101>
SYSFLAG_NEC_98 0x00000008	Server is an NEC 98 computer, which supports assignment of drive letters A and B to partitions or volumes. <102>
SYSFLAG_LAPTOP 0x00000010	Server is a laptop and does not support dynamic disks.
SYSFLAG_WOLFPACK 0x00000020	Server is running on an MSCS cluster.

clientId: Pointer to the client's object identifier.

cRemote: If set to 0, indicates that the client is on the same machine as the server. If nonzero, the client is on a different machine than the server.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

This is the first protocol message that a client sends to the server. Other protocol messages that are sent prior to this one MUST be ignored by the server.

After the server receives this message, it MUST validate the parameters.

1. Verify that *notificationInterface* is not NULL.
2. Verify that *ulIDLVersion* is not NULL.
3. Verify that *pdwFlags* is not NULL.
4. Verify that *clientId* is not NULL.
5. Verify that the client has not previously called the Initialize method.

If parameter validation fails, the server MUST immediately fail the operation, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Add the client object to the list of clients that are connected to the server.
 - Generate a new unique identifier for the client and save it in the **id** field of the client object. If the *cRemote* parameter is nonzero, the server MUST be ready to send notifications to the remote client.
 - Reference and save the pointer to the IDNotify interface that is specified by *notificationInterface* in the **notifyInterface** field of the client object. [<103>](#)

2. If successful, the identifier of the new client MUST be returned in the output parameter *clientId*.
3. If successful, the value of LDM_IDL_VERSION found in the IDL file with which the server was built MUST be returned in the output parameter *uIDLVersion*.
4. If successful, a bitmap of flags that contain information about the server MUST be returned in the output parameter *pdwFlags*.
5. Return a response to the client that contains the output parameters previously mentioned and the status of the operation.

After the client object is added to the list of clients that are connected to the server, the server MUST be ready to receive and process any protocol messages from that client and send notifications to the client.

3.2.4.4.1.56 IVolumeClient::Uninitialize (Opnum 72)

The **Uninitialize** method ends the dialogue between the client and the server.

```
HRESULT Uninitialize();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

This is the last protocol message that a client MUST send to the server. Other protocol messages sent after this one MUST be ignored by the server.

Upon receiving this message, the server MUST validate the following.

- Verify that the client object is in the list of clients currently connected to the server.

If validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Remove the client object from the list of clients currently connected to the server.
 - Dereference the pointer to the [IDMNotify](#) interface stored in the **notifyInterface** field of the client object.
2. Return a response to the client that contains the status of the operation.

The server MUST also remove the client object from the list of clients currently connected to the server if it detects that the connection to the client is lost.

3.2.4.4.1.57 IVolumeClient::Refresh (Opnum 73)

The **Refresh** method refreshes the server's cache of storage objects, including regions, removable media, CD-ROM drive media, file systems, and drive letters.

```
HRESULT Refresh();
```


This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

The server MUST process the message as follows:

1. Re-enumerate disks, disk regions, volumes, drive letters, and file systems from the system.
2. If discrepancies between the enumerated objects and the list of storage objects stored by the server are found, the server MUST make the appropriate changes to the list of storage objects and send appropriate notifications to the clients.
3. Return response to the client that contains the status of the operation.

3.2.4.4.1.58 **IVolumeClient::RescanDisks (Opnum 74)**

The **RescanDisks** method triggers detection of changes in the list of storage devices connected to the server and refreshes the server's cache of storage objects, including regions, removable media and CD-ROM drive media, file systems, drive letters, and disk drives.

```
HRESULT RescanDisks();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

The server MUST process the message as follows:

1. Ask the system to rescan all the storage buses to detect storage devices that have been connected or disconnected to and from the system.
2. Re-enumerate disks, disk regions, volumes, drive letters, and file systems from the system.
3. If discrepancies between the enumerated objects and the list of storage objects stored by the server are found, the server MUST make the appropriate changes to the list of storage objects and send appropriate notifications to the clients.
4. Return a response to the client that contains the status of the operation.

3.2.4.4.1.59 **IVolumeClient::RefreshFileSys (Opnum 75)**

The **RefreshFileSys** method refreshes the server's cache of file systems.

```
HRESULT RefreshFileSys();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

The server MUST process the message as follows:

1. Re-enumerate the file systems from the system.
2. If discrepancies between the enumerated file systems and the file systems stored in the list of storage objects are found, the server MUST make the appropriate changes to the list of storage objects and send appropriate notifications to the clients.
3. Return a response to the client containing the status of the operation.

3.2.4.4.1.60 IVolumeClient::SecureSystemPartition (Opnum 76)

The **SecureSystemPartition** method toggles the secure state of the system partition. Securing the system partition means preventing the system partition from being accessed once the system boot sequence is over. [<104>](#)

```
HRESULT SecureSystemPartition();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

The server MUST process the message as follows:

1. Toggle the secure state of the system partition (if supported by the system).
2. Return a response to the client that contains the status of the operation.

3.2.4.4.1.61 IVolumeClient::ShutDownSystem (Opnum 77)

The **ShutDownSystem** method restarts the machine on which the server is running.

```
HRESULT ShutDownSystem();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

The server MUST process the message as follows:

1. Initiate system shutdown.
2. Return a response to the client that contains the status of the operation.

If successful, the server will also be terminated as part of the system shutdown.

3.2.4.4.1.62 IVolumeClient::EnumAccessPath (Opnum 78)

The **EnumAccessPath** method enumerates all mount points configured on the server.

```

HRESULT EnumAccessPath(
    [in, out] int* lCount,
    [out, size_is(*lCount)] COUNTED_STRING** paths
);

```

lCount: The address of an **int** that returns the number of elements returned in *paths*.

paths: Pointer to an array of [COUNTED_STRING](#) structures that describe all mount points configured on the machine. Memory for the array is allocated by the server and freed by the client.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that *lCount* and *paths* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all mount points configured in the system.
2. Allocate a buffer large enough to contain **COUNTED_STRING** structures that describe all enumerated mount points.
3. Populate each **COUNTED_STRING** structure in the buffer with the mount point path.
4. The buffer MUST be returned to the client in the output parameter *paths*.
5. The number of **COUNTED_STRING** structures in the buffer MUST be returned in the output parameter *lCount*.
6. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.63 IVolumeClient::EnumAccessPathForVolume (Opnum 79)

The **EnumAccessPathForVolume** method enumerates the mount points of a specified volume, partition, or logical drive.

```

HRESULT EnumAccessPathForVolume(
    [in] LdmObjectId volumeId,
    [in, out] int* lCount,
    [out, size_is(*lCount)] COUNTED_STRING** paths
);

```

volumeId: Specifies the object identifier of the volume, partition, or logical drive for which to enumerate mount points.

lCount: The address of an **int** that returns the number of elements returned in paths.

paths: Pointer to an array of **COUNTED_STRING** structures that describe all mount points configured on the machine.

Return Values: The method MUST return 0 or a nonerror **HRESULT** on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the volume, partition, or logical drive specified by *volumeId* is in the list of storage objects.
2. Verify that *lCount* and *paths* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all mount points of the volume, partition, or logical drive specified by *volumeId*.
2. Allocate a buffer large enough to contain **COUNTED_STRING** structures describing all enumerated mount points.
3. Populate each **COUNTED_STRING** structure in the buffer with the mount pointpath.
4. The buffer MUST be returned to the client in the output parameter *paths*.
5. The number of **COUNTED_STRING** structures in the buffer MUST be returned in the output parameter *lCount*.
6. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.1.64 IVolumeClient::AddAccessPath (Opnum 80)

The **AddAccessPath** method adds the specified mount point to a volume, a partition, or a logical drive.

```
HRESULT AddAccessPath(  
    [in] int cch_path,  
    [in, size_is(cch_path)] WCHAR* path,  
    [in] LdmObjectId targetId  
);
```

cch_path: Length of path in characters, including null termination.

path: Null-terminated mount point path to assign to the volume targeted (see mounted folder). This is Unicode.

targetId: Specifies the object identifier of the volume, partition, or logical drive to which the new mount point is to be assigned.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the volume, partition, or logical drive specified by *targetId* is in the list of storage objects.
2. Verify that *path* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Assign the mount point specified by *path* to the volume, partition, or logical drive specified by *targetId*.
2. Wait for the operation to succeed or fail.
3. Return a response to the client that contains the status of the operation.

3.2.4.4.1.65 IVolumeClient::DeleteAccessPath (Opnum 81)

The **DeleteAccessPath** method deletes a specified mount point from a volume, a partition, or a logical drive.

```
HRESULT DeleteAccessPath(  
    [in] LdmObjectId volumeId,  
    [in] int cch_path,  
    [in, size_is(cch_path)] WCHAR* path  
);
```

volumeId: Specifies the object identifier of the volume, partition, or logical drive from which to delete the mount point.

cch_path: Length of path in characters, including null termination.

path: Null-terminated path of the mount point to delete.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the volume, partition, or logical drive specified by *volumeId* is in the list of storage objects.
2. Verify that *path* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Delete the mount point specified by *path* from the volume, partition, or logical drive specified by *volumeId*.
2. Wait for the operation to succeed or fail.
3. Return a response to the client that contains the status of the operation.

3.2.4.4.2 IVolumeClient2

This DCOM interface inherits the **IUnknown** interface. Method opnum field values start with 3; opnum values 0–2 represent the IUnknown_QueryInterface, IUnknown_AddRef, and IUnknown_Release methods, respectively, as specified in [\[MS-DCOM\]](#).

Unless otherwise specified below, all methods MUST return 0 or a nonerror **HRESULT** (as specified in [MS-ERREF]) on success, or an implementation-specific nonzero error code on failure (see section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Unless otherwise specified in this specification, client implementations of the protocol MUST NOT take any action on an error code, but rather simply return the error to the invoking application. If the return code is not an error, the client SHOULD assume that all output parameters are present and valid.

Methods in RPC Opnum Order

Method	Description
IVolumeClient2::GetMaxAdjustedFreeSpace	Opnum: 3

3.2.4.4.2.1 IVolumeClient2::GetMaxAdjustedFreeSpace (Opnum 3)

The **GetMaxAdjustedFreeSpace** method retrieves the maximum amount of free space on a disk, after adjusting for partition boundaries.

```
HRESULT GetMaxAdjustedFreeSpace (
    [in] LdmObjectId diskId,
    [out] LONGLONG* maxAdjustedFreeSpace
);
```

diskId: Specifies the object identifier of the disk to query.

maxAdjustedFreeSpace: Pointer to the maximum free space on the disk, adjusted for partition boundaries.

Return Values: The method MUST return 0 or a nonerror **HRESULT** on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the disk specified by *diskId* is in the list of storage objects.
2. Verify that *maxAdjustedFreeSpace* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Compute the maximum amount of free space in bytes that is available for allocation to new partitions and volumes. The computation MUST take into account any partition alignment rules enforced by the server.
2. The maximum amount of free space MUST be returned to the client in the output parameter *maxAdjustedFreeSpace*.
3. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.3 IVolumeClient3

This DCOM interface inherits the **IUnknown** interface. Method opnum field values start with 3; opnum values 0 through 2 represent the IUnknown_QueryInterface, IUnknown_AddRef, and IUnknown_Release methods, respectively, as specified in [\[MS-DCOM\]](#).

Methods with opnum field values 12 and 56–63 are not invoked across the network, and therefore are not included in this document.

Unless otherwise specified below, all methods MUST return 0 or a nonerror **HRESULT** (as specified in [\[MS-ERREF\]](#)) on success, or an implementation-specific nonzero error code on failure (see section [2.2.1](#) for **HRESULT** values pre-defined by the Disk Management Remote Protocol).

Unless otherwise specified in this specification, client implementations of the protocol MUST NOT take any action on an error code, but rather simply return the error to the invoking application. If the return code is not an error, the client SHOULD assume that all output parameters are present and valid. [<105>](#)

Methods in RPC Opnum Order

Method	Description
IVolumeClient3::EnumDisksEx	Opnum: 3
IVolumeClient3::EnumDiskRegionsEx	Opnum: 4
IVolumeClient3::CreatePartition	Opnum: 5
IVolumeClient3::CreatePartitionAssignAndFormat	Opnum: 6
IVolumeClient3::CreatePartitionAssignAndFormatEx	Opnum: 7
IVolumeClient3::DeletePartition	Opnum: 8
IVolumeClient3::InitializeDiskStyle	Opnum: 9
IVolumeClient3::MarkActivePartition	Opnum: 10
IVolumeClient3::Eject	Opnum: 11
Reserved_Opnum12	Opnum: 12
IVolumeClient3::FTEnumVolumes	Opnum: 13

Method	Description
<u>IVolumeClient3::FTEnumLogicalDiskMembers</u>	Opnum: 14
<u>IVolumeClient3::FTDeleteVolume</u>	Opnum: 15
<u>IVolumeClient3::FTBreakMirror</u>	Opnum: 16
<u>IVolumeClient3::FTResyncMirror</u>	Opnum: 17
<u>IVolumeClient3::FTRegenerateParityStripe</u>	Opnum: 18
<u>IVolumeClient3::FTReplaceMirrorPartition</u>	Opnum: 19
<u>IVolumeClient3::FTReplaceParityStripePartition</u>	Opnum: 20
<u>IVolumeClient3::EnumDriveLetters</u>	Opnum: 21
<u>IVolumeClient3::AssignDriveLetter</u>	Opnum: 22
<u>IVolumeClient3::FreeDriveLetter</u>	Opnum: 23
<u>IVolumeClient3::EnumLocalFileSystems</u>	Opnum: 24
<u>IVolumeClient3::GetInstalledFileSystems</u>	Opnum: 25
<u>IVolumeClient3::Format</u>	Opnum: 26
<u>IVolumeClient3::EnumVolumes</u>	Opnum: 27
<u>IVolumeClient3::EnumVolumeMembers</u>	Opnum: 28
<u>IVolumeClient3::CreateVolume</u>	Opnum: 29
<u>IVolumeClient3::CreateVolumeAssignAndFormat</u>	Opnum: 30
<u>IVolumeClient3::CreateVolumeAssignAndFormatEx</u>	Opnum: 31
<u>IVolumeClient3::GetVolumeMountName</u>	Opnum: 32
<u>IVolumeClient3::GrowVolume</u>	Opnum: 33
<u>IVolumeClient3::DeleteVolume</u>	Opnum: 34
<u>IVolumeClient3::CreatePartitionsForVolume</u>	Opnum: 35
<u>IVolumeClient3::DeletePartitionsForVolume</u>	Opnum: 36
<u>IVolumeClient3::GetMaxAdjustedFreeSpace</u>	Opnum: 37
<u>IVolumeClient3::AddMirror</u>	Opnum: 38
<u>IVolumeClient3::RemoveMirror</u>	Opnum: 39
<u>IVolumeClient3::SplitMirror</u>	Opnum: 40
<u>IVolumeClient3::InitializeDiskEx</u>	Opnum: 41
<u>IVolumeClient3::UninitializeDisk</u>	Opnum: 42

Method	Description
<u>IVolumeClient3::ReConnectDisk</u>	Opnum: 43
<u>IVolumeClient3::ImportDiskGroup</u>	Opnum: 44
<u>IVolumeClient3::DiskMergeQuery</u>	Opnum: 45
<u>IVolumeClient3::DiskMerge</u>	Opnum: 46
<u>IVolumeClient3::ReAttachDisk</u>	Opnum: 47
<u>IVolumeClient3::ReplaceRaid5Column</u>	Opnum: 48
<u>IVolumeClient3::RestartVolume</u>	Opnum: 49
<u>IVolumeClient3::GetEncapsulateDiskInfoEx</u>	Opnum: 50
<u>IVolumeClient3::EncapsulateDiskEx</u>	Opnum: 51
<u>IVolumeClient3::QueryChangePartitionNumbers</u>	Opnum: 52
<u>IVolumeClient3::DeletePartitionNumberInfoFromRegistry</u>	Opnum: 53
<u>IVolumeClient3::SetDontShow</u>	Opnum: 54
<u>IVolumeClient3::GetDontShow</u>	Opnum: 55
Reserved0	Opnum: 56
Reserved1	Opnum: 57
Reserved2	Opnum: 58
Reserved3	Opnum: 59
Reserved4	Opnum: 60
Reserved5	Opnum: 61
Reserved6	Opnum: 62
Reserved7	Opnum: 63
<u>IVolumeClient3::EnumTasks</u>	Opnum: 64
<u>IVolumeClient3::GetTaskDetail</u>	Opnum: 65
<u>IVolumeClient3::AbortTask</u>	Opnum: 66
<u>IVolumeClient3::HrGetErrorData</u>	Opnum: 67
<u>IVolumeClient3::Initialize</u>	Opnum: 68
<u>IVolumeClient3::Uninitialize</u>	Opnum: 69
<u>IVolumeClient3::Refresh</u>	Opnum: 70
<u>IVolumeClient3::RescanDisks</u>	Opnum: 71

Method	Description
IVolumeClient3::RefreshFileSys	Opnum: 72
IVolumeClient3::SecureSystemPartition	Opnum: 73
IVolumeClient3::ShutDownSystem	Opnum: 74
IVolumeClient3::EnumAccessPath	Opnum: 75
IVolumeClient3::EnumAccessPathForVolume	Opnum: 76
IVolumeClient3::AddAccessPath	Opnum: 77
IVolumeClient3::DeleteAccessPath	Opnum: 78

3.2.4.4.3.1 IVolumeClient3::EnumDisksEx (Opnum 3)

The **EnumDisksEx** method enumerates the server's mass storage devices.

```
HRESULT EnumDisksEx(
    [out] unsigned long* diskCount,
    [out, size_is(*diskCount)] DISK_INFO_EX** diskList
);
```

diskCount: Pointer to the number of elements in *diskList*.

diskList: Pointer to an array of [DISK_INFO_EX](#) structures.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

- Verify that *diskCount* and *diskList* are not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all disk objects from the list of storage objects.
2. Allocate a buffer large enough to contain **DISK_INFO_EX** structures that describe all enumerated disks.
3. Populate each **DISK_INFO_EX** structure in the buffer with information about the disk.
4. The buffer MUST be returned to the client in the output parameter *diskList*.
5. The number of **DISK_INFO_EX** structures in the buffer MUST be returned in the output parameter *diskCount*.
6. Return a response containing the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.3.2 IVolumeClient3::EnumDiskRegionsEx (Opnum 4)

The **EnumDiskRegionsEx** method enumerates all used and free regions of a specified disk.

```
HRESULT EnumDiskRegionsEx(  
    [in] LdmObjectId diskId,  
    [in, out] unsigned long* numRegions,  
    [out, size_is(*numRegions)] REGION_INFO_EX** regionList  
);
```

diskId: Specifies the object identifier of the disk for which regions are being enumerated.

numRegions: Pointer to the number of regions in *regionList*.

regionList: Pointer to an array of [REGION_INFO_EX](#) structures.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that *numRegions* and *regionList* are not NULL.
2. Verify that the disk specified by *diskId* is in the list of storage objects.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Enumerate all disk region objects that reside on the specified disk.
2. Allocate a buffer large enough to contain **REGION_INFO_EX** structures that describing all regions that reside on the disk.
3. The buffer MUST be populated with regions in the ascending order of the byte offset of the region relative to the beginning of the disk.
4. Populate each **REGION_INFO_EX** structure in the buffer with information about the region.
5. The buffer MUST be returned to the client in the output parameter *regionList*.
6. The number of **REGION_INFO_EX** structures in the buffer MUST be returned in the output parameter *numRegions*.
7. Return a response to the client containing the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.3.3 IVolumeClient3::CreatePartition (Opnum 5)

The **CreatePartition** method creates a partition.

```

HRESULT CreatePartition(
    [in] REGION_SPEC partitionSpec,
    [out] TASK_INFO* tinfo
);

```

partitionSpec: A [REGION_SPEC](#) structure that defines the partition type and length to create.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::CreatePartition** message, it MUST process that message, as specified in [3.2.4.4.1.3](#).

3.2.4.4.3.4 IVolumeClient3::CreatePartitionAssignAndFormat (Opnum 6)

The **CreatePartitionAssignAndFormat** method creates a partition, formats it as a file system, and assigns it a drive letter.

```

HRESULT CreatePartitionAssignAndFormat(
    [in] REGION_SPEC partitionSpec,
    [in] wchar_t letter,
    [in] hyper letterLastKnownState,
    [in] FILE_SYSTEM_INFO fsSpec,
    [in] boolean quickFormat,
    [out] TASK_INFO* tinfo
);

```

partitionSpec: A [REGION_SPEC](#) structure that defines the type and length of the partition to create.

letter: Drive letter to assign to the new volume.

letterLastKnownState: Drive letter's last known modification sequence number. This value is returned from a call to EnumDriveLetters.

fsSpec: A [FILE_SYSTEM_INFO](#) structure that defines the file system to create.

quickFormat: Boolean value that determines whether the server will fully format or quickly format the file system.

Value	Meaning
FALSE 0	File system will be fully formatted. Full format requires verifying the accessibility of all sectors on the volume.
TRUE 1	File system will be quickly formatted.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::CreatePartitionAssignAndFormat** message, it MUST process that message, as specified in [IVolumeClient::CreatePartitionAssignAndFormat \(section 3.2.4.4.1.4\)](#).

3.2.4.4.3.5 IVolumeClient3::CreatePartitionAssignAndFormatEx (Opnum 7)

The **CreatePartitionAssignAndFormatEx** method creates a partition, formats it as a file system, and assigns it a drive letter and a mount point.

```
HRESULT CreatePartitionAssignAndFormatEx(  
    [in] REGION_SPEC partitionSpec,  
    [in] wchar_t letter,  
    [in] hyper letterLastKnownState,  
    [in] int cchAccessPath,  
    [in, size_is(,cchAccessPath)] wchar_t* AccessPath,  
    [in] FILE_SYSTEM_INFO fsSpec,  
    [in] boolean quickFormat,  
    [in] DWORD dwFlags,  
    [out] TASK_INFO* tinfo  
);
```

partitionSpec: A [REGION_SPEC](#) structure that defines the type and length of the partition to create.

letter: Drive letter to assign to the new volume.

letterLastKnownState: Drive letter's last known modification sequence number.

cchAccessPath: Length of the *AccessPath* parameter, in Unicode characters, including null termination.

AccessPath: Null-terminated Unicode string that specifies the path in which the new file system is being mounted. This parameter is used to supply a mounted folder path for the case where the new partition will be mounted to a directory on another volume.

fsSpec: A [FILE_SYSTEM_INFO](#) structure that defines the file system to create. This parameter is returned from a call to EnumLocalFileSystems().

quickFormat: Value that indicates whether the server will fully format or quickly format the file system.

Value	Meaning
FALSE 0	File system will be fully formatted. Full format requires verifying the accessibility of all sectors on the volume.
TRUE 1	File system will be quickly formatted.

dwFlags: Bitmap of partition creation flags.

Value	Meaning
CREATE_ASSIGN_ACCESS_PATH 0x00000001	Assign the mount point <i>AccessPath</i> to the new partition.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::CreatePartitionAssignAndFormatEx** message, it MUST process that message, as specified in [IVolumeClient::CreatePartitionAssignAndFormatEx \(section 3.2.4.4.1.5\)](#).

3.2.4.4.3.6 IVolumeClient3::DeletePartition (Opnum 8)

The **DeletePartition** method deletes a specified partition.

```
HRESULT DeletePartition(
    [in] REGION_SPEC partitionSpec,
    [in] boolean force,
    [out] TASK_INFO* tinfo
);
```

partitionSpec: A [REGION_SPEC](#) structure that specifies the type and length of the partition to delete.

force: Value that determines if deletion of the partition will be forced. If the force parameter is not set, the call will fail if the volume cannot be locked.

Value	Meaning
FALSE 0	Deletion will not be forced if the partition is in use.
TRUE 1	Deletion will be forced.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::DeletePartition** message, it MUST process that message, as specified in [IVolumeClient::DeletePartition \(section 3.2.4.4.1.6\)](#).

3.2.4.4.3.7 IVolumeClient3::InitializeDiskStyle (Opnum 9)

The **InitializeDiskStyle** method sets the partition style and writes a signature to a disk. This is a synchronous task.

```
HRESULT InitializeDiskStyle(
```

```

[in] LdmObjectId diskId,
[in] PARTITIONSTYLE style,
[in] hyper diskLastKnownState,
[out] TASK_INFO* tinfo
);

```

diskId: Specifies the object identifier of the target disk for the signature.

style: Value from the [PARTITIONSTYLE](#) enumeration that indicates the partition style to use.

diskLastKnownState: Last known modification sequence number of the disk.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the disk specified by *diskId* is in the list of storage objects, and check if *diskLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Initialize the disk specified by *diskId* with an empty partition table and write a signature to it.
 - If style is PARTITIONSTYLE_MBR, the disk is initialized with an MBR partition table and signature.
 - If style is PARTITIONSTYLE_GPT, the disk is initialized with a GPT partition table and signature.
2. Wait for the initialization to either succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <106>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.

TASK_INFO member	Required for this operation
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <107>

4. Return response to the client containing *tinfo* and the status of the operation.

If the operation is successful, the server MUST make the following change to the list of storage objects before returning the response.

- Modify the disk object to account for the change of status.

3.2.4.4.3.8 IVolumeClient3::MarkActivePartition (Opnum 10)

The **MarkActivePartition** method marks a specified partition as the active partition of the disk.

```
HRESULT MarkActivePartition(
    [in] LdmObjectId regionId,
    [in] hyper regionLastKnownState,
    [out] TASK_INFO* tinfo
);
```

regionId: Specifies the object identifier of the partition to activate.

regionLastKnownState: Partition's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::MarkActivePartition** message, it MUST process that message, as specified in [IVolumeClient::MarkActivePartition \(section 3.2.4.4.1.8\)](#).

3.2.4.4.3.9 IVolumeClient3::Eject (Opnum 11)

The **Eject** method ejects a specified removable disk or CD-ROM from the drive enclosure.

```
HRESULT Eject(
    [in] LdmObjectId diskId,
    [in] hyper diskLastKnownState,
    [out] TASK_INFO* tinfo
);
```

diskId: Specifies the object identifier of the media to eject.

diskLastKnownState: Disk's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::Eject** message, it MUST process that message, as specified in [IVolumeClient::Eject \(section 3.2.4.4.1.9\)](#).

3.2.4.4.3.10 IVolumeClient3::FTEnumVolumes (Opnum 13)

The **FTEnumVolumes** method enumerates the server's fault-tolerant volumes on basic disks (rather than dynamic).

```
HRESULT FTEnumVolumes(  
    [in, out] unsigned long* volumeCount,  
    [out, size_is(*volumeCount)] VOLUME_INFO** ftVolumeList  
);
```

volumeCount: Pointer to the number of elements in *ftVolumeList*.

ftVolumeList: Pointer to an array of [VOLUME_INFO](#) structures.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::FTEnumVolumes** message, it MUST process that message, as specified in [IVolumeClient::FTEnumVolumes \(section 3.2.4.4.1.10\)](#).

3.2.4.4.3.11 IVolumeClient3::FTEnumLogicalDiskMembers (Opnum 14)

The **FTEnumLogicalDiskMembers** method enumerates the regions of a specified fault tolerant volume on basic disks (rather than dynamic).

```
HRESULT FTEnumLogicalDiskMembers(  
    [in] LdmObjectId volumeId,  
    [in, out] unsigned long* memberCount,  
    [out, size_is(*memberCount)] LdmObjectId** memberList  
);
```

volumeId: Specifies the object identifier of the volume for which regions are being enumerated.

memberCount: Pointer to the number of regions that the volume includes. The client passes in the address of an unsigned long.

memberList: Pointer to an array of [LdmObjectId](#) objects that store member identification handles for the regions in the volume.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::FTEnumLogicalDiskMembers** message, it MUST process that message, as specified in [IVolumeClient::FTEnumLogicalDiskMembers \(section 3.2.4.4.1.11\)](#).

3.2.4.4.3.12 IVolumeClient3::FTDeleteVolume (Opnum 15)

The **FTDeleteVolume** method deletes the fault tolerant volume specified by *volumeId* on a basic disk (rather than a dynamic disk).

```
HRESULT FTDeleteVolume(  
    [in] LdmObjectId volumeId,  
    [in] boolean force,  
    [in] hyper volumeLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the volume to delete.

force: Boolean value that indicates if deletion of a partition will be forced. The call to delete will fail if the volume is locked by some other application and this flag is not set.

Value	Meaning
FALSE 0	Deletion will not be forced if the partition is in use.
TRUE 1	Deletion of the partition will be forced.

volumeLastKnownState: Volume's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::FTDeleteVolume** message, it MUST process that message, as specified in [IVolumeClient3::FTDeleteVolume \(section 3.2.4.4.1.12\)](#)

3.2.4.4.3.13 IVolumeClient3::FTBreakMirror (Opnum 16)

The **FTBreakMirror** method breaks a specified fault-tolerant mirror set into two independent partitions.

```
HRESULT FTBreakMirror(  
    [in] LdmObjectId volumeId,  
    [in] hyper volumeLastKnownState,  
    [in] boolean bForce,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the FT mirror set to break.

volumeLastKnownState: Last known modification sequence number of the FT mirror set.

bForce: Boolean value that indicates whether to force removal of the drive letter from the FT mirror set.

Value	Meaning
FALSE 0	The method fails if an error occurs while the drive letter is being removed from the FT mirror set.
TRUE 1	Removal of the drive letter from the FT mirror set is forced.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::FTBreakMirror** message, it MUST process that message, as specified in [IVolumeClient::FTBreakMirror \(section 3.2.4.4.1.13\)](#).

3.2.4.4.3.14 IVolumeClient3::FTResyncMirror (Opnum 17)

The **FTResyncMirror** method restores the redundancy of an FT mirror set by resynchronizing the members of the mirror.

```
HRESULT FTResyncMirror(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the FT mirror set being resynchronized.

volumeLastKnownState: Last known modification sequence number of the FT mirror set.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::FTResyncMirror** message, it MUST process that message, as specified in [IVolumeClient::FTResyncMirror \(section 3.2.4.4.1.14\)](#).

3.2.4.4.3.15 IVolumeClient3::FTRegenerateParityStripe (Opnum 18)

The **FTRegenerateParityStripe** method restores the redundancy of an FT RAID-5 set by regenerating the parity of the volume.

```
HRESULT FTRegenerateParityStripe(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the FT RAID-5 set for which the parity is being regenerated.

volumeLastKnownState: Last known modification sequence number of the FT RAID-5 set.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::FTRegenerateParityStripe** message, it MUST process that message, as specified in [IVolumeClient::FTRegenerateParityStripe \(section 3.2.4.4.1.15\)](#).

3.2.4.4.3.16 IVolumeClient3::FTReplaceMirrorPartition (Opnum 19)

The **FTReplaceMirrorPartition** method repairs an FT mirror set by replacing the failed member of the set with another partition. This method operates on a fault tolerant volume on a basic disk (rather than a dynamic disk). The partition MUST have the same type as the original, it MUST be MBR, and it MUST be at least as big as the original partition.

```
HRESULT FTReplaceMirrorPartition(  
    [in] LdmObjectId volumeId,  
    [in] hyper volumeLastKnownState,  
    [in] LdmObjectId oldMemberId,  
    [in] hyper oldMemberLastKnownState,  
    [in] LdmObjectId newRegionId,  
    [in] hyper newRegionLastKnownState,  
    [in] DWORD flags,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the FT mirror set to modify.

volumeLastKnownState: FT mirror set's last known modification sequence number.

oldMemberId: This parameter MUST be set to 0 by the client and ignored by the server.

oldMemberLastKnownState: This parameter MUST be set to 0 by the client and ignored by the server.

newRegionId: Specifies the object identifier of the replacement partition. The partition MUST have the same type as the original, it MUST be MBR, and it MUST be at least as big as the original partition.

newRegionLastKnownState: Last known modification sequence number of the replacement partition.

flags: Bitmap of flags for the replacement operation. The value of this field is a logical 'OR' of zero or more of the applicable flags defined below.

Value	Meaning
FTREPLACE_FORCE 0x00000001	Do not fail the operation if the replacement partition has been changed since <i>newRegionLastKnownState</i> .
FTREPLACE_DELETE_ON_FAIL 0x00000002	Delete the replacement partition if operation fails.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::FTReplaceMirrorPartition** message, it MUST process that message, as specified in [IVolumeClient3::FTReplaceMirrorPartition \(section 3.2.4.4.1.16\)](#).

3.2.4.4.3.17 IVolumeClient3::FTReplaceParityStripePartition (Opnum 20)

The **FTReplaceParityStripePartition** method repairs an FT RAID-5 set by replacing the failed member of the set with another partition. The partition MUST have the same type as the original, it MUST be MBR, and it MUST be at least as big as the original partition.

```
HRESULT FTReplaceParityStripePartition(  
    [in] LdmObjectId volumeId,  
    [in] hyper volumeLastKnownState,  
    [in] LdmObjectId oldMemberId,  
    [in] hyper oldMemberLastKnownState,  
    [in] LdmObjectId newRegionId,  
    [in] hyper newRegionLastKnownState,  
    [in] DWORD flags,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the FT RAID-5 set to modify.

volumeLastKnownState: Last known modification sequence number of the FT RAID-5 set.

oldMemberId: This parameter MUST be set to 0 by the client and ignored by the server.

oldMemberLastKnownState: This parameter MUST be set to 0 by the client and ignored by the server.

newRegionId: Specifies the object identifier of the replacement partition. The partition MUST have the same type as the original, it MUST be MBR, and it MUST be at least as big as the original partition.

newRegionLastKnownState: Last known modification sequence number of the replacement partition.

flags: Bitmap of flags for the replacement operation.

Value	Meaning
FTREPLACE_FORCE 0x00000001	Do not fail the operation if the replacement partition has been changed since <i>newRegionLastKnownState</i> .
FTREPLACE_DELETE_ON_FAIL 0x00000002	Delete the replacement partition if operation fails.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::FTReplaceParityStripePartition** message, it MUST process that message as specified in [IVolumeClient::FTReplaceParityStripePartition \(section 3.2.4.4.1.17\)](#).

3.2.4.4.3.18 IVolumeClient3::EnumDriveLetters (Opnum 21)

The **EnumDriveLetters** method enumerates the server's drive letters, both used and free. For drive letters that are in use, the method returns the mapping between the drive letter and the volume, partition, or logical drive using it.

```
HRESULT EnumDriveLetters(  
    [in, out] unsigned long* driveLetterCount,  
    [out, size_is(*driveLetterCount)]  
        DRIVE_LETTER_INFO** driveLetterList  
);
```

driveLetterCount: Pointer to the number of elements returned in *driveLetterList*.

driveLetterList: Pointer to an array of [DRIVE_LETTER_INFO](#) structures.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::EnumDriveLetters** message, it MUST process that message, as specified in [IVolumeClient::EnumDriveLetters \(section 3.2.4.4.1.18\)](#).

3.2.4.4.3.19 IVolumeClient3::AssignDriveLetter (Opnum 22)

The **AssignDriveLetter** method assigns the specified drive letter to a volume, partition, or logical drive.

```
HRESULT AssignDriveLetter(  
    [in] wchar_t letter,  
    [in] unsigned long forceOption,  
    [in] hyper letterLastKnownState,  
    [in] LdmObjectId storageId,  
    [in] hyper storageLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

letter: Drive letter to assign, specified as a single case-insensitive Unicode character.

forceOption: Value that indicates if drive letter assignment is forced when it fails. This method call will fail if the force flag is not set and some other application has a lock on the volume.

Value	Meaning
NO_FORCE_OPERATION	If the volume, partition, or logical drive specified by <i>storageId</i> already has a drive letter assigned, and freeing it fails because the object is in

Value	Meaning
0x00000000	use, assignment fails and the old drive letter is retained.
FORCE_OPERATION 0x00000001	If the volume, partition, or logical drive specified by <i>storageId</i> already has a drive letter assigned, and freeing it fails because the volume is in use, its removal is forced and assignment of the new drive letter succeeds.

letterLastKnownState: Drive letter's last known modification sequence number.

storageId: Specifies the object identifier of the volume, partition, or logical drive to which the drive letter is being assigned.

storageLastKnownState: Last known modification sequence number of the volume, partition, or logical drive to which the drive letter is being assigned.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::AssignDriveLetter** message, it MUST process that message as specified in Section [IVolumeClient3::AssignDriveLetter \(section 3.2.4.4.1.19\)](#).

3.2.4.4.3.20 IVolumeClient3::FreeDriveLetter (Opnum 23)

The **FreeDriveLetter** method unassigns a specified drive letter from a volume, partition, or logical drive on the server.

```
HRESULT FreeDriveLetter(
    [in] wchar_t letter,
    [in] unsigned long forceOption,
    [in] hyper letterLastKnownState,
    [in] LdmObjectId storageId,
    [in] hyper storageLastKnownState,
    [out] TASK_INFO* tinfo
);
```

letter: Drive letter to free.

forceOption: Boolean value that indicates whether to force the freeing of a drive letter. This call will fail if some other application has the volume locked.

Value	Meaning
NO_FORCE_OPERATION 0	If the specified drive letter is assigned to a volume, partition, or logical disk that is in use, contains the paging file, or contains the system directory, the operation fails and returns an error.
FORCE_OPERATION 1	The specified drive letter is always freed.

letterLastKnownState: Drive letter's last known modification sequence number.

storageId: Specifies the object identifier of the volume, partition, or logical drive to which the letter is assigned.

storageLastKnownState: Last known modification sequence number of the volume, partition, or logical drive to which the drive letter is assigned.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::FreeDriveLetter** message, it MUST process that message, as specified in [IVolumeClient::FreeDriveLetter \(section 3.2.4.4.1.20\)](#).

3.2.4.4.3.21 IVolumeClient3::EnumLocalFileSystems (Opnum 24)

The **EnumLocalFileSystems** method enumerates the file systems present on the server.

```
HRESULT EnumLocalFileSystems (
    [out] unsigned long* fileSystemCount,
    [out, size_is(*fileSystemCount)]
    FILE_SYSTEM_INFO** fileSystemList
);
```

fileSystemCount: Pointer to the number of elements returned in *fileSystemList*. The client passes in the address of an unsigned long.

fileSystemList: Pointer to an array of [FILE_SYSTEM_INFO](#) structures that represent the file systems present on the server.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::EnumLocalFileSystems** message, it MUST process that message, as specified in [IVolumeClient::EnumLocalFileSystems \(section 3.2.4.4.1.21\)](#).

3.2.4.4.3.22 IVolumeClient3::GetInstalledFileSystems (Opnum 25)

The **GetInstalledFileSystems** method enumerates the file system types (for example, FAT or NTFS) that the server supports.

```
HRESULT GetInstalledFileSystems (
    [out] unsigned long* fsCount,
    [out, size_is(*fsCount)] IFILE_SYSTEM_INFO** fsList
);
```

fsCount: Pointer to the number of elements returned in *fsList*.

fsList: Pointer to an array of [IFILE_SYSTEM_INFO](#) structures.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::GetInstalledFileSystems** message, it MUST process that message, as specified in [IVolumeClient::GetInstalledFileSystems \(section 3.2.4.4.1.22\)](#).

3.2.4.4.3.23 IVolumeClient3::Format (Opnum 26)

The **Format** method formats the specified volume, partition, or logical drive with a file system.

```
HRESULT Format(  
    [in] LdmObjectId storageId,  
    [in] FILE_SYSTEM_INFO fsSpec,  
    [in] boolean quickFormat,  
    [in] boolean force,  
    [in] hyper storageLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

storageId: Specifies the object identifier of the volume, partition, or logical drive on which the new file system is being created.

fsSpec: A [FILE_SYSTEM_INFO](#) structure that specifies details about the file system being created.

quickFormat: Boolean value that indicates if the file system will be fully formatted. This call will fail if this flag is not set and some other application has the volume locked.

Value	Meaning
FALSE 0	File system will be fully formatted. Full format requires verifying the accessibility of all sectors on the volume.
TRUE 1	File system will be quickly formatted.

force: Boolean value that indicates if the file system will be formatted if the volume, partition, or logical drive cannot be locked.

Value	Meaning
FALSE 0	File system will not be formatted unless its underlying storage can be locked.
TRUE 1	File system will be formatted regardless of whether the underlying volume, partition, or logical drive can be locked or not.

storageLastKnownState: Last known modification sequence number of the volume, partition, or logical drive on which the file system is being created.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::Format** message, it MUST process that message, as specified in [IVolumeClient::Format \(section 3.2.4.4.1.23\)](#).

3.2.4.4.3.24 IVolumeClient3::EnumVolumes (Opnum 27)

The **EnumVolumes** method enumerates the dynamic volumes of the server.

```
HRESULT EnumVolumes(  
    [in, out] unsigned long* volumeCount,  
    [out, size_is(*volumeCount)] VOLUME_INFO** LdmVolumeList  
);
```

volumeCount: Pointer to the number of elements returned in *LdmVolumeList*.

LdmVolumeList: Pointer to an array of [VOLUME_INFO](#) structures representing the dynamic volumes of the server.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::EnumVolumes** message, it MUST process that message, as specified in [IVolumeClient::EnumVolumes \(section 3.2.4.4.1.24\)](#).

3.2.4.4.3.25 IVolumeClient3::EnumVolumeMembers (Opnum 28)

The **EnumVolumeMembers** method enumerates the regions of the specified dynamic volume.

```
HRESULT EnumVolumeMembers(  
    [in] LdmObjectId volumeId,  
    [in, out] unsigned long* memberCount,  
    [out, size_is(*memberCount)] LdmObjectId** memberList  
);
```

volumeId: Specifies the object identifier of the volume for which regions are being enumerated.

memberCount: Pointer to the number of disk regions returned in *memberList*.

memberList: Array of [LdmObjectId](#) objects that store the identification handles of the regions.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::EnumVolumeMembers** message, it MUST process that message, as specified in [IVolumeClient::EnumVolumeMembers \(section 3.2.4.4.1.25\)](#).

3.2.4.4.3.26 IVolumeClient3::CreateVolume (Opnum 29)

The **CreateVolume** method creates a dynamic volume on the specified list of disks.

```

HRESULT CreateVolume(
    [in] VOLUME_SPEC volumeSpec,
    [in] unsigned long diskCount,
    [in, size_is(diskCount)] DISK_SPEC* diskList,
    [out] TASK_INFO* tinfo
);

```

volumeSpec: A [VOLUME_SPEC](#) structure that defines the parameters of the volume to create.

diskCount: Number of elements passed in *diskList*.

diskList: Array of [DISK_SPEC](#) structures that specifies the disks to be used by the volume.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::CreateVolume** message, it MUST process that message, as specified in [IVolumeClient3::CreateVolume \(section 3.2.4.4.1.26\)](#).

3.2.4.4.3.27 IVolumeClient3::CreateVolumeAssignAndFormat (Opnum 30)

The **CreateVolumeAssignAndFormat** method creates a dynamic volume on the specified list of disks, assigns a drive letter to it, and formats it with a file system.

```

HRESULT CreateVolumeAssignAndFormat(
    [in] VOLUME_SPEC volumeSpec,
    [in] unsigned long diskCount,
    [in, size_is(diskCount)] DISK_SPEC* diskList,
    [in] wchar_t letter,
    [in] hyper letterLastKnownState,
    [in] FILE_SYSTEM_INFO fsSpec,
    [in] boolean quickFormat,
    [out] TASK_INFO* tinfo
);

```

volumeSpec: A [VOLUME_SPEC](#) structure that defines the volume to create.

diskCount: Number of elements passed in *diskList*.

diskList: Array of [DISK_SPEC](#) structures that specifies the disks to be used by the volume.

letter: Drive letter to assign to the new volume. If no drive letter is needed for the volume, the value of this field MUST be a two-byte NULL character or the Unicode SPACE character.

letterLastKnownState: Drive letter's last known modification sequence number.

fsSpec: A [FILE_SYSTEM_INFO](#) structure that defines the file system to create.

quickFormat: Value that indicates whether the server will fully format or quickly format the file system.

Value	Meaning
FALSE 0	File system will be fully formatted. Full format requires verifying the accessibility of all sectors on the volume.
TRUE 1	File system will be quickly formatted.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::CreateVolumeAssignAndFormat** message, it MUST process that message, as specified in [IVolumeClient::CreateVolumeAssignAndFormat \(section 3.2.4.4.1.27\)](#).

3.2.4.4.3.28 IVolumeClient3::CreateVolumeAssignAndFormatEx (Opnum 31)

The **CreateVolumeAssignAndFormatEx** method creates a dynamic volume on the specified list of disks, assigns a drive letter and/or a mount point to it, and formats it with a file system.

```
HRESULT CreateVolumeAssignAndFormatEx(
    [in] VOLUME_SPEC volumeSpec,
    [in] unsigned long diskCount,
    [in, size_is(diskCount)] DISK_SPEC* diskList,
    [in] wchar_t letter,
    [in] hyper letterLastKnownState,
    [in] int cchAccessPath,
    [in, size_is(cchAccessPath)] wchar_t* AccessPath,
    [in] FILE_SYSTEM_INFO fsSpec,
    [in] boolean quickFormat,
    [in] DWORD dwFlags,
    [out] TASK_INFO* tinfo
);
```

volumeSpec: A [VOLUME_SPEC](#) structure that defines the volume to create.

diskCount: Number of elements passed in *diskList*.

diskList: Array of [DISK_SPEC](#) structures that specifies the disk to be used by the volume. Memory for the array is allocated and freed by the client.

letter: Drive letter to assign to the new volume. Pass the zero value or the SPACE character if no drive letter is needed.

letterLastKnownState: Drive letter's last known modification sequence number.

cchAccessPath: Length of *AccessPath* including null termination.

AccessPath: Null-terminated path in which the new file system is being mounted. The server MUST ignore this parameter if the CREATE_ASSIGN_ACCESS_PATH bit is not set in *dwFlags*.

fsSpec: A [FILE_SYSTEM_INFO](#) structure that defines the file system to create.

quickFormat: Value that indicates whether the server will fully format or quickly format the file system.

Value	Meaning
FALSE 0	File system will be fully formatted. Full format requires verifying the accessibility of all sectors on the volume.
TRUE 1	File system will be quickly formatted.

dwFlags: Bitmap of volume creation flags. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
CREATE_ASSIGN_ACCESS_PATH 0x00000001	Assign the mount point <i>AccessPath</i> to the new volume. If the flag is not set, the parameter <i>AccessPath</i> is ignored.

info: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::CreateVolumeAssignAndFormatEx** message, it MUST process that message, as specified in [IVolumeClient::CreateVolumeAssignAndFormatEx \(section 3.2.4.4.1.28\)](#).

3.2.4.4.3.29 IVolumeClient3::GetVolumeMountName (Opnum 32)

The **GetVolumeMountName** method retrieves the mount name for a volume, partition, or logical drive.

```
HRESULT GetVolumeMountName(  
    [in] LdmObjectId volumeId,  
    [out] unsigned long* cchMountName,  
    [out, size_is(*cchMountName)]  
    WCHAR** mountName  
);
```

volumeId: Specifies the object identifier of the volume for which the mount name is being retrieved.

cchMountName: Pointer to the length of *mountName*, including null termination.

mountName: Pointer to the null-terminated mount name of the volume in the format \\?\Volume{guid}(note that the question mark is literal, not a wildcard).

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::GetVolumeMountName** message, it MUST process that message, as specified in [IVolumeClient::GetVolumeMountName \(section 3.2.4.4.1.29\)](#).

3.2.4.4.3.30 IVolumeClient3::GrowVolume (Opnum 33)

The **GrowVolume** method increases the length of a specified dynamic volume by appending extents from the specified disks to it.

```
HRESULT GrowVolume(  
    [in] LdmObjectId volumeId,  
    [in] VOLUME_SPEC volumeSpec,  
    [in] unsigned long diskCount,  
    [in, size_is(diskCount)] DISK_SPEC* diskList,  
    [in] boolean force,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the volume whose size is being changed.

volumeSpec: A [VOLUME_SPEC](#) structure that defines the parameters of the changed volume, including its new expected length.

diskCount: Number of elements passed in *diskList*.

diskList: Array of [DISK_SPEC](#) structures that specifies the list of disk extents to be appended to the volume.

force: Boolean value that determines whether the volume is extended or not, in case it cannot be locked.

Value	Meaning
FALSE 0	Volume is not extended unless it is locked.
TRUE 1	Volume is extended whether it is locked or unlocked.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **VolumeClient3::GrowVolumeI** message, it MUST process that message, as specified in [IVolumeClient::GrowVolume \(section 3.2.4.4.1.30\)](#).

3.2.4.4.3.31 IVolumeClient3::DeleteVolume (Opnum 34)

The **DeleteVolume** method deletes the specified dynamic volume.

```
HRESULT DeleteVolume(  
    [in] LdmObjectId volumeId,  
    [in] boolean force,  
    [in] hyper volumeLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the volume to delete.

force: Value that indicates if deletion of the volume will be forced if the volume is in use. If this value is false, the call will fail if some other application has the volume locked.

Value	Meaning
FALSE 0	Deletion will not be forced if the volume is in use.
TRUE 1	Deletion will be forced.

volumeLastKnownState: Volume's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::DeleteVolume** message, it MUST process that message, as specified in [IVolumeClient::DeleteVolume \(section 3.2.4.4.1.31\)](#).

3.2.4.4.3.32 IVolumeClient3::CreatePartitionsForVolume (Opnum 35)

The **CreatePartitionsForVolume** method creates a partition underneath a volume. This is a synchronous task.

```
HRESULT CreatePartitionsForVolume(  
    [in] LdmObjectId volumeId,  
    [in] boolean active,  
    [in] hyper volumeLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the volume under which to create a partition.

active: Boolean value that indicates if the new partition is to be set to active, which would make it an active partition. On x86, and possibly other BIOSes, this is needed by the BIOS to start the machine from the volume.

Value	Meaning
FALSE 0	New partition is not set to active.
TRUE 1	New partition is set to active.

volumeLastKnownState: Last known modification sequence number of the volume.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects, and check if the field **volumeSpec.lastKnownState** matches the field **LastKnownState** of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Enumerate all disk regions that correspond to the dynamic volume specified by *volumeId* from the list of storage objects.
2. For each disk region, create an entry in the partition table of its disk. The partition MUST have the same offset and length as the disk region. If the active flag is set to TRUE, set the active bit in the partition table to 1. If the active flag is set to FALSE, set the active bit in the partition table to 0.
3. Wait for the partition creations to succeed or fail.
4. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. .<108>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. .<109>

5. Return a response to the client containing *tinfo* and the status of the operation.
6. Send the task completion notification.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

- Modify the dynamic volume object to account for the change of status.

- Modify the disk objects where the partitions were created to account for the change in region allocation.

3.2.4.4.33 IVolumeClient3::DeletePartitionsForVolume (Opnum 36)

The **DeletePartitionsForVolume** method deletes the partitions underneath a dynamic diskvolume. This is a synchronous task.

```
HRESULT DeletePartitionsForVolume(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the object identifier of the volume under which to delete partitions.

volumeLastKnownState: Last known modification sequence number of the volume.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects, and check if the field **volumeSpec.lastKnownState** matches the field **LastKnownState** of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Enumerate all disk regions that correspond to the dynamic volume specified by *volumeId* from the list of storage objects.
2. For each disk region, delete the entry in the partition table of its disk that has the same offset and length as the disk region.
3. Wait for the partition deletions to succeed or fail.
4. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <110>
TASK_INFO::clientId	Required.

TASK_INFO member	Required for this operation
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <111>

5. Return a response to the client containing *volumeId* and the status of the operation.
6. Send the task completion notification.

If the operation is successful, the server MUST make the following changes to the list of storage objects before returning the response.

- Modify the dynamic volume object to account for the change of status.
- Modify the disk objects where the partitions were deleted to account for the change in region allocation.

3.2.4.4.3.34 IVolumeClient3::GetMaxAdjustedFreeSpace (Opnum 37)

The **GetMaxAdjustedFreeSpace** method retrieves the maximum amount of free space on a disk after adjusting for partition boundaries.

```
HRESULT GetMaxAdjustedFreeSpace(
    [in] LdmObjectId diskId,
    [out] LONGLONG* maxAdjustedFreeSpace
);
```

diskId: Specifies the object identifier of the disk to query.

maxAdjustedFreeSpace: Pointer to the maximum free space on the disk, adjusted for partition boundaries.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::GetMaxAdjustedFreeSpace** message, it MUST process that message, as specified in [IVolumeClient2::GetMaxAdjustedFreeSpace \(section 3.2.4.4.2.1\)](#).

3.2.4.4.3.35 IVolumeClient3::AddMirror (Opnum 38)

The **AddMirror** method adds a mirror to the specified dynamic volume.

```
HRESULT AddMirror(
    [in] LdmObjectId volumeId,
    [in] hyper volumeLastKnownState,
    [in] DISK_SPEC diskSpec,
```

```

[in, out] int* diskNumber,
[out] int* partitionNumber,
[out] TASK_INFO* tinfo
);

```

volumeId: Specifies the object identifier of the volume to which the mirror is being added.

volumeLastKnownState: Volume's last known modification sequence number.

diskSpec: A [DISK_SPEC](#) structure that defines the disk to add as a mirror.

diskNumber: This parameter MUST be set to 0 by the client and MUST be ignored by the server.

partitionNumber: If the *volumeId* parameter is the boot volume, this parameter returns a pointer to the partition number of the newly added mirror. If the volume is not the boot volume, the server MUST return partition number zero.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::AddMirror** message, it MUST process that message, as specified in [IVolumeClient::AddMirror \(section 3.2.4.4.1.32\)](#).

3.2.4.4.3.36 IVolumeClient3::RemoveMirror (Opnum 39)

The **RemoveMirror** method removes a mirror from a dynamic volume.

```

HRESULT RemoveMirror(
[in] LdmObjectId volumeId,
[in] hyper volumeLastKnownState,
[in] LdmObjectId diskId,
[in] hyper diskLastKnownState,
[out] TASK_INFO* tinfo
);

```

volumeId: Specifies the object identifier of the mirrored volume from which the disk is being removed.

volumeLastKnownState: Volume's last known modification sequence number.

diskId: Specifies the object identifier of the disk being removed from the volume.

diskLastKnownState: Last known modification sequence number of the disk being removed from the volume.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::RemoveMirror** message, it MUST process that message, as specified in [IVolumeClient::RemoveMirror \(section 3.2.4.4.1.33\)](#).

3.2.4.4.3.37 IVolumeClient3::SplitMirror (Opnum 40)

The **SplitMirror** method splits a dynamic mirrored volume into two independent simple volumes. One of them keeps the identifier and drive letter of the original volume while the other is assigned a different identity.

```
HRESULT SplitMirror(  
    [in] LdmObjectId volumeId,  
    [in] hyper volumeLastKnownState,  
    [in] LdmObjectId diskId,  
    [in] hyper diskLastKnownState,  
    [in] wchar_t letter,  
    [in] hyper letterLastKnownState,  
    [in, out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the volume to split.

volumeLastKnownState: Volume's last known modification sequence number.

diskId: Specifies the object identifier of the disk to split off from the volume specified by *volumeId*.

diskLastKnownState: Last known modification sequence number of the disk to split off.

letter: Drive letter to assign to the disk identified by *diskId*. If no drive letter is needed for the volume, the value of this field MUST be a two-byte Unicode NULL character or the Unicode SPACE character.

letterLastKnownState: Last known modification sequence number of the drive letter that is being assigned to the disk to split.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::SplitMirror** message, it MUST process that message, as specified in [IVolumeClient::SplitMirror \(section 3.2.4.4.1.34\)](#).

3.2.4.4.3.38 IVolumeClient3::InitializeDiskEx (Opnum 41)

The **InitializeDiskEx** method initializes a disk for control by the volume manager. This is a synchronous task.

```
HRESULT InitializeDiskEx(  
    [in] LdmObjectId diskId,  
    [in] PARTITIONSTYLE style,  
    [in] hyper diskLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

diskId: Specifies the object identifier of the disk to initialize for volume manager control.

style: Value from the PARTITIONSTYLE enumeration that indicates the partition style to use.

diskLastKnownState: Last known modification sequence number of the disk.

tinfo: Pointer to a [TASK_INFO](#) structure the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that the disk specified by *diskId* is in the list of storage objects, and check if *diskLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Initialize the disk specified by *diskId* with an empty partition table and write a signature to it:
 1. If style is PARTITIONSTYLE_MBR, the disk is initialized with an MBR partition table and signature.
 2. If style is PARTITIONSTYLE_GPT, the disk is initialized with a GPT partition table and signature.
2. If successful, convert the disk to a dynamic disk.
3. Wait for the conversion to succeed or fail.
4. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <112>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <113>

5. Return a response to the client containing *tinfo* and the status of the operation.

6. Send the task completion notification.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

- Modify the disk object to account for the change in type.
- Delete disk region objects that reside on the uninitialized disk.
- Create disk region objects that reside on the dynamic disk.

3.2.4.4.3.39 **IVolumeClient3::UninitializeDisk (Opnum 42)**

The **UninitializeDisk** method removes a disk from control by the volume manager.

```
HRESULT UninitializeDisk(  
    [in] LdmObjectId diskId,  
    [in] hyper diskLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

diskId: Specifies the object identifier of the disk to remove from volume manager control.

diskLastKnownState: Last known modification sequence number of the disk.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::UninitializeDisk** message, it **MUST** process that message, as specified in [IVolumeClient::UninitializeDisk \(section 3.2.4.4.1.36\)](#).

3.2.4.4.3.40 **IVolumeClient3::ReConnectDisk (Opnum 43)**

The **ReConnectDisk** method reactivates a failed dynamic disk, bringing the disk and the volumes that reside on it online.

```
HRESULT ReConnectDisk(  
    [in] LdmObjectId diskId,  
    [out] TASK_INFO* tinfo  
);
```

diskId: Specifies the object identifier of the disk to reactivate.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::ReConnectDisk** message, it **MUST** process that message, as specified in [IVolumeClient::ReConnectDisk \(section 3.2.4.4.1.37\)](#).

3.2.4.4.3.41 IVolumeClient3::ImportDiskGroup (Opnum 44)

The **ImportDiskGroup** method imports a foreign dynamic disk group as the primary disk group of the server.

```
HRESULT ImportDiskGroup(  
    [in] int cchDgid,  
    [in, size_is(cchDgid)] byte* dgid,  
    [out] TASK_INFO* tinfo  
);
```

cchDgid: Size of *dgid* in characters, including null termination.

dgid: Null-terminated ASCII string that contains the universally unique identifier (UUID) of the disk group to import.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::ImportDiskGroup** message, it MUST process that message, as specified in [IVolumeClient::ImportDiskGroup \(section 3.2.4.4.1.38\)](#).

3.2.4.4.3.42 IVolumeClient3::DiskMergeQuery (Opnum 45)

The **DiskMergeQuery** method gathers disk and volume information needed to merge a foreign dynamic disk group into the primary disk group of the server.

```
HRESULT DiskMergeQuery(  
    [in] int cchDgid,  
    [in, size_is(cchDgid)] byte* dgid,  
    [in] int numDisks,  
    [in, size_is(numDisks)] LdmObjectId* diskList,  
    [out] hyper* merge_config_tid,  
    [out] int* numRids,  
    [out, size_is(*numRids)] hyper** merge_dm_rids,  
    [out] int* numObjects,  
    [out, size_is(*numObjects)] MERGE_OBJECT_INFO** mergeObjectInfo,  
    [in, out] unsigned long* flags,  
    [out] TASK_INFO* tinfo  
);
```

cchDgid: Size of *dgid* in characters, including null termination.

dgid: Null-terminated ASCII string that contains the universally unique identifier (UUID) of the disk group to be merged.

numDisks: Number of disks passed in *diskList*.

diskList: Array of object identifiers of type [LdmObjectId](#) that specify the disks of the *dgid* group to be merged.

merge_config_tid: Pointer to the modification sequence number of the disk group to be merged.

numRids: Pointer to the number of elements returned in *merge_dm_rids*.

merge_dm_rids: Pointer to an array of disk records that represent the disks that will be merged. Memory for the array is allocated by the server and freed by the client.

numObjects: Number of elements returned in *mergeObjectInfo*.

mergeObjectInfo: Pointer to an array of [MERGE_OBJECT_INFO](#) structures that contain information about the volumes that will be merged.

flags: Disk merge query flags. The value of this field is a logical 'OR' of zero or more of the applicable flags defined below.

Value	Meaning
DSKMERGE_IN_NO_UNRELATED 0x00000001	Do not retrieve merge information for volumes of the foreigndisk group that do not have extents on <i>diskList</i> . This is an input-only flag.
DSKMERGE_OUT_NO_PRIMARY_DG 0x00000001	The machine does not have a primary disk group. This is an output-only flag.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::DiskMergeQuery** message, it MUST process that message, as specified in [IVolumeClient3::DiskMergeQuery \(section 3.2.4.4.1.39\)](#).

3.2.4.4.3.43 IVolumeClient3::DiskMerge (Opnum 46)

The **DiskMerge** method merges a foreign disk group into the primary disk group of the server. The foreign disks and their volumes are brought online.

```
HRESULT DiskMerge(  
    [in] int cchDgid,  
    [in, size_is(cchDgid)] byte* dgid,  
    [in] int numDisks,  
    [in, size_is(numDisks)] LdmObjectId* diskList,  
    [in] hyper merge_config_tid,  
    [in] int numRids,  
    [in, size_is(numRids)] hyper* merge_dm_rids,  
    [out] TASK_INFO* tinfo  
);
```

cchDgid: Size of *dgid* in characters, including null termination.

dgid: Null-terminated ASCII string that contains the universally unique identifier (UUID) of the disk group to be merged.

numDisks: Number of disks passed in *diskList*.

diskList: Array of object identifiers of type [LdmObjectId](#) that specify the disks to be merged from the *dgid* group.

merge_config_tid: Last known modification sequence number of the disk group to be merged.

numRids: Number of elements passed in *merge_dm_rids*.

merge_dm_rids: Array of disk records for the disks in *diskList*.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::DiskMerge** message, it MUST process that message, as specified in [IVolumeClient::DiskMerge \(section 3.2.4.4.1.40\)](#).

3.2.4.4.3.44 IVolumeClient3::ReAttachDisk (Opnum 47)

The **ReAttachDisk** method reattaches the specified dynamic disk, bringing the volumes of the disk online after reconnecting the disk device to the server.

```
HRESULT ReAttachDisk(  
    [in] LdmObjectId diskId,  
    [in] hyper diskLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

diskId: Specifies the object identifier of the disk to reattach.

diskLastKnownState: Disk's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::ReAttachDisk** message, it MUST process that message, as specified in [IVolumeClient::ReAttachDisk \(section 3.2.4.4.1.41\)](#).

3.2.4.4.3.45 IVolumeClient3::ReplaceRaid5Column (Opnum 48)

The **ReplaceRaid5Column** method repairs a dynamic RAID-5 volume by replacing the failed member of the volume with a specified disk.

```
HRESULT ReplaceRaid5Column(  
    [in] LdmObjectId volumeId,  
    [in] hyper volumeLastKnownState,  
    [in] LdmObjectId newDiskId,  
    [in] hyper diskLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the volume for which the member will be replaced.

volumeLastKnownState: Last known modification sequence number of the RAID-5 volume.

newDiskId: Specifies the object identifier of the replacement disk.

diskLastKnownState: Replacement disk's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::ReplaceRaid5Column** message, it MUST process that message, as specified in [IVolumeClient::ReplaceRaid5Column \(section 3.2.4.4.1.43\)](#).

3.2.4.4.3.46 IVolumeClient3::RestartVolume (Opnum 49)

The **RestartVolume** method attempts to bring a dynamic volume back online.

```
HRESULT RestartVolume(  
    [in] LdmObjectId volumeId,  
    [in] hyper volumeLastKnownState,  
    [out] TASK_INFO* tinfo  
);
```

volumeId: Specifies the object identifier of the volume to restart.

volumeLastKnownState: Volume's last known modification sequence number.

tinfo: Pointer to a [TASK_INFO](#) structure that the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::RestartVolume** message, it MUST process that message, as specified in [IVolumeClient::RestartVolume \(section 3.2.4.4.1.44\)](#).

3.2.4.4.3.47 IVolumeClient3::GetEncapsulateDiskInfoEx (Opnum 50)

The **GetEncapsulateDiskInfoEx** method gathers the information needed to convert the specified basic disks to dynamic disks.

```
HRESULT GetEncapsulateDiskInfoEx(  
    [in] unsigned long diskCount,  
    [in, size_is(diskCount)] DISK_SPEC* diskSpecList,  
    [out] unsigned long* encapInfoFlags,  
    [out] unsigned long* affectedDiskCount,  
    [out, size_is(*affectedDiskCount)]  
    DISK_INFO_EX** affectedDiskList,  
    [out, size_is(*affectedDiskCount)]  
    unsigned long** affectedDiskFlags,  
    [out] unsigned long* affectedVolumeCount,  
    [out, size_is(*affectedVolumeCount)]
```

```

    VOLUME_INFO** affectedVolumeList,
[out] unsigned long* affectedRegionCount,
[out, size_is(*affectedRegionCount)]
    REGION_INFO_EX** affectedRegionList,
[out] TASK_INFO* tinfo
);

```

diskCount: Number of elements passed in the *diskSpecList* array.

diskSpecList: Array of [DISK_SPEC](#) structures that specify the disks to be encapsulated. Memory for the array is allocated and freed by the client.

encapInfoFlags: Bitmap of flags that returns information about encapsulating the disks specified in *diskSpecList*. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
ENCAP_INFO_CANT_PROCEED 0x00000001	Encapsulation for the disk will not succeed. The other flags specify the reason.
ENCAP_INFO_NO_FREE_SPACE 0x00000002	Volume manager could not find sufficient free space on the disk for encapsulation.
ENCAP_INFO_BAD_ACTIVE 0x00000004	Disk contains an active partition from which the current operating system was not started.
ENCAP_INFO_UNKNOWN_PART 0x00000008	Volume manager was unable to determine the type of a partition on the disk because of corruption or other errors reading the disk. For example, any error that prevents the partition information from being read, the partition is neither GPT nor MBR, an OEM partition is found that is not at the beginning of the disk.
ENCAP_INFO_FT_UNHEALTHY 0x00000010	Disk contains an FT set volume that is not functioning properly.
ENCAP_INFO_FT_QUERY_FAILED 0x00000020	Volume manager was unable to obtain information about an FT set volume on the disk.
ENCAP_INFO_REBOOT_REQD 0x00000100	Encapsulation of the disk requires a restart of the computer.
ENCAP_INFO_CONTAINS_FT 0x00000200	Disk is part of an FT set volume.
ENCAP_INFO_VOLUME_BUSY 0x00000400	Disk is currently in use.
ENCAP_INFO_PART_NR_CHANGE 0x00000800	Encapsulation of the disk requires modification of the boot configuration.
ENCAP_INFO_MIXED_PARTITIONS 0x00001000	Encapsulation of a GPT disk that contains basic partitions mixed with non-basic partitions is not supported.
ENCAP_INFO_OPEN_FAILED 0x00002000	Could not open a volume that resides on a disk in the set of disks specified for encapsulation.

affectedDiskCount: Pointer to the number of disks that will be affected by the encapsulation.

affectedDiskList: Pointer to an array of new [DISK_INFO_EX](#) structures that represent the disks that will be affected by the encapsulation.

affectedDiskFlags: Pointer to an array of bitmaps of flags that provides information about the disks that will be affected by the encapsulation. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
CONTAINS_FT 0x00000001	Disk contains an FT set volume.
CONTAINS_RAID5 0x00000002	Disk contains part of an FT RAID-5 set.
CONTAINS_REDISTRIBUTION 0x00000004	Not used.
CONTAINS_BOOTABLE_PARTITION 0x00000008	Disk contains a bootable partition.
CONTAINS_LOCKED_PARTITION 0x00000010	Disk contains a locked partition.
CONTAINS_NO_FREE_SPACE 0x00000020	Disk is full.
CONTAINS_EXTENDED_PARTITION 0x00000040	Disk contains an extended partition.
PARTITION_NUMBER_CHANGE 0x00000080	A partition number on the disk has changed.
CONTAINS_BOOTINDICATOR 0x00000100	Disk contains the active partition.
CONTAINS_BOOTLOADER 0x00000200	Disk contains the boot loader.
CONTAINS_SYSTEMDIR 0x00000400	Partition contains the system directory.
CONTAINS_MIXED_PARTITIONS 0x00000800	Partition contains partitions that will not be converted to dynamic.

affectedVolumeCount: Pointer to the number of volumes that will be affected by the encapsulation.

affectedVolumeList: Pointer to an array of [VOLUME_INFO](#) structures that represent the volumes that will be affected by the encapsulation.

affectedRegionCount: Pointer to the number of regions that will be affected by the encapsulation.

affectedRegionList: Pointer to an array of [REGION_INFO_EX](#) structures that represent the regions that will be affected by the encapsulation.

tinfo: Pointer to a [TASK_INFO](#) structure the client can use to track the request's progress.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that *diskCount* is not 0 and *diskSpecList* is not NULL.
2. For each **DISK_SPEC** structure specified in *diskSpecList*, verify that the disk specified by *diskId* is in the list of storage objects; and check if *lastKnownState* matches the **LastKnownState** field of the object.
3. Verify that *encapInfoFlags* is not NULL.
4. Verify that *affectedDiskCount*, *affectedDiskList*, and *affectedDiskFlags* are not NULL.
5. Verify that *affectedVolumeCount* and *affectedVolumeList* are not NULL.
6. Verify that *affectedRegionCount* and *affectedRegionList* are not NULL.
7. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Identify other basic disks from the list of storage objects that need to be encapsulated together with the basic disks specified by *diskSpecList*.
2. Allocate a buffer large enough to contain **DISK_INFO_EX** structures that describe all basic disks that need to be encapsulated together (including the disks specified by *diskSpecList*).
3. Populate each **DISK_INFO_EX** structure in the buffer with information about the disk.
4. The buffer MUST be returned to the client in the output parameter *affectedDiskList*.
5. The number of **DISK_INFO_EX** structures in the buffer MUST be returned to the client in the output parameter *affectedDiskCount*.
6. Allocate a second buffer large enough to contain bitmaps of flags, one for each disk returned in *affectedDiskList*, that describes disk conditions that are of interest to clients in the context of encapsulation.
7. Populate the second buffer with the bitmaps of flags of the disks.
8. The second buffer MUST be returned to the client in the output parameter *affectedDiskFlags*. Note that the list of elements in the buffer is the same as the value returned in *affectedDiskCount*.
9. Enumerate all the FT volumes that reside on the disks returned in *affectedDiskList* from the list of storage objects.
10. Allocate a third buffer large enough to contain **VOLUME_INFO** structures that describe the enumerated FT volumes.

11. Populate each **VOLUME_INFO** structure in the third buffer with information about the FT volume.
12. The third buffer MUST be returned to the client in the output parameter *affectedVolumeList*.
13. The number of **VOLUME_INFO** structures in the third buffer MUST be returned to the client in the output parameter *affectedVolumeCount*.
14. Enumerate all the disk regions that reside on the disks returned in *affectedDiskList* from the list of storage objects.
15. Allocate a fourth buffer large enough to contain **REGION_INFO_EX** structures that describe the enumerated disk regions.
16. Populate each **REGION_INFO_EX** structure in the fourth buffer with information about the disk region.
17. The fourth buffer MUST be returned to the client in the output parameter *affectedRegionList*.
18. The number of **REGION_INFO_EX** structures in the fourth buffer MUST be returned to the client in the output parameter *affectedRegionCount*.
19. Populate a 32-bit-signed integer bitmap of flags that describes conditions that will prevent the overall encapsulation to proceed, or might be of interest to the client in the context of encapsulation. If the encapsulation cannot proceed, the server MUST set the **ENCAP_INFO_CANT_PROCEED** flag, and then set other flags as appropriate to account for the reasons why the encapsulation is not possible.
20. The bitmap of flags MUST be returned to the client in the output parameter *encapInfoFlags*.
21. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.3.48 IVolumeClient3::EncapsulateDiskEx (Opnum 51)

The **EncapsulateDiskEx** method converts the specified basic disks to dynamic disks. This is a synchronous task.

```
HRESULT EncapsulateDiskEx(
    [in] unsigned long affectedDiskCount,
    [in, size_is(affectedDiskCount)]
        DISK_INFO_EX* affectedDiskList,
    [in] unsigned long affectedVolumeCount,
    [in, size_is(affectedVolumeCount)]
        VOLUME_INFO* affectedVolumeList,
    [in] unsigned long affectedRegionCount,
    [in, size_is(affectedRegionCount)]
        REGION_INFO_EX* affectedRegionList,
    [out] unsigned long* encapInfoFlags,
    [out] TASK_INFO* tinfo
);
```

affectedDiskCount: The number of elements passed in the *affectedDiskList* array.

affectedDiskList: An array of [DISK_INFO_EX](#) structures that specify the disks to be encapsulated.

affectedVolumeCount: The number of elements passed in the *affectedVolumeList* array.

affectedVolumeList: An array of [VOLUME_INFO](#) structures that represent the basic volumes affected by the encapsulation.

affectedRegionCount: The number of elements passed in the *affectedRegionList* array.

affectedRegionList: An array of [REGION_INFO_EX](#) structures that represent the regions affected by the encapsulation.

encapInfoFlags: Bitmap of flags that provides information about the encapsulation. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
ENCAP_INFO_CANT_PROCEED 0x00000001	Encapsulation for disk did not succeed. Inspect the other values of <i>encapInfoFlags</i> to determine the reason.
ENCAP_INFO_NO_FREE_SPACE 0x00000002	The volume manager could not find sufficient free space on the disk for encapsulation.
ENCAP_INFO_BAD_ACTIVE 0x00000004	The disk contains an active partition from which the current operating system was not started.
ENCAP_INFO_UNKNOWN_PART 0x00000008	The volume manager was unable to determine the type of a partition on the disk.
ENCAP_INFO_FT_UNHEALTHY 0x00000010	The disk contains an unhealthy FT set volume.
ENCAP_INFO_FT_QUERY_FAILED 0x00000020	The volume manager was unable to obtain information about an FT set volume on the disk.
ENCAP_INFO_REBOOT_REQD 0x00000100	Encapsulation of the disk will require a restart of the computer.
ENCAP_INFO_CONTAINS_FT 0x00000200	The disk is part of an FT set volume.
ENCAP_INFO_VOLUME_BUSY 0x00000400	The disk is currently in use.
ENCAP_INFO_PART_NR_CHANGE 0x00000800	Encapsulation of the disk requires modification of the boot configuration.
ENCAP_INFO_MIXED_PARTITIONS 0x00001000	Encapsulation of a GPTdisk that contains basic partitions mixed with non-basic partitions is not supported.
ENCAP_INFO_OPEN_FAILED 0x00002000	Could not open a volume that resides on a disk in the set of disks specified for encapsulation.

tinfo: A pointer to a [TASK_INFO](#) structure that the client can use to track the progress of the request.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that *affectedDiskList* is not NULL and *affectedDiskCount* is not 0.
2. For each **DISK_INFO_EX** structure specified by *affectedDiskList*, verify that the disk specified by *diskId* is in the list of storage objects and *lastKnownState* matches the **LastKnownState** field of the object.
3. Verify that no other basic disks need to be encapsulated together with the disks specified by *affectedDiskList*.
4. Verify that *affectedVolumeList* is NULL and *affectedVolumeCount* is 0.
5. Verify that the list of basic volumes specified by *affectedVolumeList* matches the set of basic volumes residing on the disks specified by *affectedDiskList*.
6. For each **REGION_INFO_EX** structure specified by *affectedRegionList*, verify that the disk region specified by *regionId* is in the list of storage objects and *lastKnownState* matches the **LastKnownState** field of the object.
7. Verify that the list of disk regions specified by *affectedRegionList* matches the set of disk regions residing on the disks specified by *affectedDiskList*.
8. Verify that *encapInfoFlags* is not NULL.
9. Verify that *tinfo* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST process the message as follows:

1. Convert the basic disks specified by *affectedDiskList* to dynamic.
 - All partitions and logical drives that reside on the basic disk are converted to dynamic volumes.
2. Wait for the conversion to succeed or fail.
3. Fill in the *tinfo* output parameter.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required. <114>
TASK_INFO::clientID	Required.
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.

TASK_INFO member	Required for this operation
TASK_INFO::type	Required if PercentageComplete is being used.
TASK_INFO::error	Required.
TASK_INFO::tflag	Not required. <115>

4. Return a response to the client containing *info* and the status of the operation.
5. Send the task completion notification.

If the operation is successful, the server **MUST** make the following changes to the list of storage objects before returning the response.

1. Modify the converted disk objects to account for the change in type.
2. Create new dynamic volume objects that correspond to the new dynamic volumes.
3. Create new disk region objects for the new dynamic disks.
4. Delete disk region objects of the old basic disks.
5. Modify drive letter objects to account for the change of volume owning them.
6. Modify file system objects to account for the change of volume owning them.

If the boot partition is among the disks being encapsulated, the server **MUST** store boot partition change information on persistent storage (registry). The information **MUST** contain the old (pre-encapsulation) and new (post-encapsulation) partition number of the boot partition. The information is useful in case the client sends an [IVolumeClient3::QueryChangePartitionNumbers](#) message.

3.2.4.4.3.49 IVolumeClient3::QueryChangePartitionNumbers (Opnum 52)

The **QueryChangePartitionNumbers** method retrieves information about the partition number change that results when a boot partition is encapsulated.

```
HRESULT QueryChangePartitionNumbers(
    [out] int* oldPartitionNumber,
    [out] int* newPartitionNumber
);
```

oldPartitionNumber: Pointer to the partition number of the boot volume before the encapsulation operation.

newPartitionNumber: Pointer to the partition number of the boot volume after the encapsulation operation.

Return Values: The method **MUST** return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::QueryChangePartitionNumbers** message, it **MUST** process that message, as specified in [IVolumeClient::QueryChangePartitionNumbers \(section 3.2.4.4.1.47\)](#).

3.2.4.4.3.50 IVolumeClient3::DeletePartitionNumberInfoFromRegistry (Opnum 53)

The **DeletePartitionNumberInfoFromRegistry** method deletes the boot partition number change history from persistent storage.

```
HRESULT DeletePartitionNumberInfoFromRegistry();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::DeletePartitionNumberInfoFromRegistry** message, it MUST process that message, as specified in [IVolumeClient::DeletePartitionNumberInfoFromRegistry \(section 3.2.4.4.1.48\)](#).

3.2.4.4.3.51 IVolumeClient3::SetDontShow (Opnum 54)

See Windows behavior tag.[<116>](#)

```
HRESULT SetDontShow(  
    [in] boolean bSetNoShow  
);
```

bSetNoShow: Boolean value that determines if the new Disk Wizard is enabled.

Value	Meaning
FALSE 0	Enable new Disk Wizard. This is the default value. This value indicates that the user has not selected the check box in the New Disk Wizard to request that the wizard not to be displayed in the future.
TRUE 1	Disable new Disk Wizard. This value indicates that the user has selected the check box in the New Disk Wizard to request that the wizard not to be displayed in the future.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::SetDontShow** message, it MUST process that message, as specified in [IVolumeClient::SetDontShow \(section 3.2.4.4.1.49\)](#).

3.2.4.4.3.52 IVolumeClient3::GetDontShow (Opnum 55)

See the Windows behavior tag.[<117>](#)

```
HRESULT GetDontShow(  
    [out] boolean* bGetNoShow  
);
```

bGetNoShow: Boolean value that indicates if the New Disk Wizard is enabled or disabled.

Value	Meaning
FALSE 0	New Disk Wizard is enabled. This is the default value. This value indicates that the user has not selected the check box in the New Disk Wizard to request that the wizard not to be displayed in the future.
TRUE 1	New Disk Wizard is disabled. Indicates that the user has selected the check box in the New Disk Wizard to request that the wizard not to be displayed in the future.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::GetDontShow** message, it MUST process that message, as specified in [IVolumeClient::GetDontShow \(section 3.2.4.4.1.50\)](#).

3.2.4.4.3.53 IVolumeClient3::EnumTasks (Opnum 64)

The **EnumTasks** method enumerates the tasks currently running on the server.

```
HRESULT EnumTasks(  
    [in, out] unsigned long* taskCount,  
    [out, size_is(*taskCount)] TASK_INFO** taskList  
);
```

taskCount: Number of elements returned in the *taskList* array.

taskList: Array of [TASK_INFO](#) structures that describes the tasks running on the server.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::EnumTasks** message, it MUST process that message, as specified in [IVolumeClient::EnumTasks \(section 3.2.4.4.1.51\)](#).

3.2.4.4.3.54 IVolumeClient3::GetTaskDetail (Opnum 65)

The **GetTaskDetail** method retrieves information about a task running on the server.

```
HRESULT GetTaskDetail(  
    [in] IdmObjectId id,  
    [in, out] TASK_INFO* tinfo  
);
```

id: Specifies the object identifier of the task for which to retrieve information.

tinfo: A [TASK_INFO](#) structure that describes the operation currently being performed by *id*.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::GetTaskDetail** message, it MUST process that message, as specified in [IVolumeClient::GetTaskDetail \(section 3.2.4.4.1.52\)](#).

3.2.4.4.3.55 IVolumeClient3::AbortTask (Opnum 66)

The **AbortTask** method aborts a task running on the server.

```
HRESULT AbortTask(  
    [in] LdmObjectId id  
);
```

id: Specifies the object identifier of the task to be aborted.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::AbortTask** message, it MUST process that message, as specified in [IVolumeClient::AbortTask \(section 3.2.4.4.1.53\)](#).

3.2.4.4.3.56 IVolumeClient3::HrGetErrorData (Opnum 67)

The **HrGetErrorData** method retrieves user-readable error information associated with an [HRESULT](#) error code.

```
HRESULT HrGetErrorData(  
    [in] HRESULT hr,  
    [in] DWORD dwFlags,  
    [out] DWORD* pdwStoredFlags,  
    [out] int* pcszw,  
    [out, string, size_is(*pcszw)]  
    wchar_t*** prgszw  
);
```

hr: The HRESULT error code from which error information is retrieved.

dwFlags: Bitmap of retrieval flags. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
ERRFLAG_NOREMOVE 0x00020000	Do not delete the error information from the list maintained by the server.
ERRFLAG_IGNORETAG 0x00040000	Retrieve the error information even if it was not produced for this client.

pdwStoredFlags: Pointer to a bitmap of error flags. [<118>](#)

pcszw: Pointer to the number of strings returned in *prgszw*.

prgszw: Pointer to an array of strings that contain error information for the HRESULT.

Return Values: The method MUST return 0 or a nonerror **HRESULT** on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::HrGetErrorData** message, it MUST process that message, as specified in [IVolumeClient::HrGetErrorData \(section 3.2.4.4.1.54\)](#).

3.2.4.4.3.57 IVolumeClient3::Initialize (Opnum 68)

The **Initialize** method initializes the dialogue between the client and the server. This method MUST be the first call made by the client after connecting to the server.

```
HRESULT Initialize(  
    [in] IUnknown* notificationInterface,  
    [out] unsigned long* ulIDLVersion,  
    [out] DWORD* pdwFlags,  
    [out] LdmObjectId* clientId,  
    [in] unsigned long cRemote  
);
```

notificationInterface: Pointer to the client's **IUnknown** interface from which the server can query the [IDMNotify](#) interface used for sending notifications to the client.

ulIDLVersion: Revision of the **Microsoft Interface Definition Language (MIDL)** file with which the server was built.

pdwFlags: Bitmap of information flags about the server. The value of this field is generated by combining zero or more of the applicable flags defined below with a logical OR operation.

Value	Meaning
SYSFLAG_SERVER 0x00000001	Server is running on Windows Server 2003 and Windows 2000 Server.
SYSFLAG_ALPHA 0x00000002	Server is running on an Alpha processor. <119>
SYSFLAG_SYSPART_SECURE 0x00000004	System partition for the server is secure. <120>
SYSFLAG_NEC_98 0x00000008	Server is an NEC 98 computer, which supports assignment of drive letters A and B to partitions or volumes. <121>
SYSFLAG_NO_DYNAMIC 0x00000010	Server is a laptop and does not support dynamic disks.
SYSFLAG_WOLFPACK 0x00000020	Server is running on an MSCS cluster.
SYSFLAG_IA64 0x00000040	Server is running on an Intel IA64 processor.
SYSFLAG_UNINSTALL_VALID 0x00000080	Server has an available and valid backup for uninstall.
SYSFLAG_DYNAMIC_1394	Server supports converting IEEE 1394 attached disks to dynamic

Value	Meaning
0x00000100	disks.

clientId: Pointer to the client's object identifier.

cRemote: If set to 0, indicates that the client is on the same machine as the server. Otherwise, the client is on a different machine than the server.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::Initialize** message, it MUST process that message, as specified in [IVolumeClient::Initialize \(section 3.2.4.4.1.55\)](#).

3.2.4.4.3.58 IVolumeClient3::Uninitialize (Opnum 69)

The **Uninitialize** method ends the dialogue between the client and the server.

```
HRESULT Uninitialize();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::Uninitialize** message, it MUST process that message, as specified in [IVolumeClient::Uninitialize \(section 3.2.4.4.1.56\)](#).

3.2.4.4.3.59 IVolumeClient3::Refresh (Opnum 70)

The **Refresh** method refreshes the server's cache of storage objects, including regions, removable media and CD-ROM drive media, file systems, and drive letters.

```
HRESULT Refresh();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::Refresh** message, it MUST process that message, as specified in [IVolumeClient::Refresh \(section 3.2.4.4.1.57\)](#).

3.2.4.4.3.60 IVolumeClient3::RescanDisks (Opnum 71)

The **RescanDisks** method triggers the detection of changes in the list of storage devices connected to the server and refreshes the server's cache of storage objects, including regions, removable media and CD-ROM drive media, file systems, drive letters, and disk drives.

```
HRESULT RescanDisks();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::RescanDisks** message, it MUST process that message, as specified in [IVolumeClient::RescanDisks \(section 3.2.4.4.1.58\)](#).

3.2.4.4.3.61 IVolumeClient3::RefreshFileSys (Opnum 72)

The **RefreshFileSys** method refreshes the server's cache of file systems.

```
HRESULT RefreshFileSys();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::RefreshFileSys** message, it MUST process that message, as specified in [IVolumeClient::RefreshFileSys \(section 3.2.4.4.1.59\)](#).

3.2.4.4.3.62 IVolumeClient3::SecureSystemPartition (Opnum 73)

The **SecureSystemPartition** method toggles the secure state of the system partition. Securing the system partition means preventing the system partition from being accessed once the system boot sequence is over.

```
HRESULT SecureSystemPartition();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::SecureSystemPartition** message, it MUST process that message, as specified in [IVolumeClient::SecureSystemPartition \(section 3.2.4.4.1.60\)](#).

3.2.4.4.3.63 IVolumeClient3::ShutDownSystem (Opnum 74)

The **ShutDownSystem** method restarts the machine on which the server is running.

```
HRESULT ShutDownSystem();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::ShutDownSystem** message, it MUST process that message, as specified in [IVolumeClient::ShutDownSystem \(section 3.2.4.4.1.61\)](#).

3.2.4.4.3.64 IVolumeClient3::EnumAccessPath (Opnum 75)

The **EnumAccessPath** method enumerates all mount points configured on the machine.

```
HRESULT EnumAccessPath(  
    [in, out] int* lCount,  
    [out, size_is(*lCount)] COUNTED_STRING** paths  
);
```

lCount: The address of an int that returns the number of elements returned in paths.

paths: Pointer to an array of [COUNTED_STRING](#) structures that describes all mount points configured on the machine.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::EnumAccessPath** message, it MUST process that message, as specified in [IVolumeClient::EnumAccessPath \(section 3.2.4.4.1.62\)](#).

3.2.4.4.3.65 IVolumeClient3::EnumAccessPathForVolume (Opnum 76)

The **EnumAccessPathForVolume** method enumerates the mount points of a specified volume, partition, or logical drive.

```
HRESULT EnumAccessPathForVolume(  
    [in] IdmObjectId volumeId,  
    [in, out] int* lCount,  
    [out, size_is(*lCount)] COUNTED_STRING** paths  
);
```

volumeId: Specifies the object identifier of the volume, partition, or logical drive for which to enumerate mount points.

lCount: The address of an int that returns the number of elements returned in paths.

paths: Pointer to an array of [COUNTED_STRING](#) structures that describe the volume's mount points.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::EnumAccessPathForVolume** message, it MUST process that message, as specified in [IVolumeClient::EnumAccessPathForVolume \(section 3.2.4.4.1.63\)](#).

3.2.4.4.3.66 IVolumeClient3::AddAccessPath (Opnum 77)

The **AddAccessPath** method adds the specified mount point to a volume, partition, or logical drive.

```
HRESULT AddAccessPath(  
    [in] int cch_path,  
    [in, size_is(cch_path)] WCHAR* path,  
    [in] LdmObjectId targetId  
);
```

cch_path: Length of path in characters, including null termination.

path: Null-terminated mount point path to assign to the volume *targetId*.

targetId: Specifies the object identifier of the volume, partition, or logical drive to which the new mount point is to be assigned.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::AddAccessPath** message, it MUST process that message, as specified in [IVolumeClient::AddAccessPath \(section 3.2.4.4.1.64\)](#).

3.2.4.4.3.67 IVolumeClient3::DeleteAccessPath (Opnum 78)

The **DeleteAccessPath** method deletes a specified mount point from a volume, partition, or logical drive.

```
HRESULT DeleteAccessPath(  
    [in] LdmObjectId volumeId,  
    [in] int cch_path,  
    [in, size_is(cch_path)] WCHAR* path  
);
```

volumeId: Specifies the object identifier of the volume, partition, or logical drive from which to delete the mount point.

cch_path: Length of path in characters, including null termination.

path: Null-terminated path of the mount point to delete.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

When the server receives an **IVolumeClient3::DeleteAccessPath** message, it MUST process that message, as specified in [IVolumeClient::DeleteAccessPath \(section 3.2.4.4.1.65\)](#).

3.2.4.4.4 IVolumeClient4

Unless otherwise specified below, all methods MUST return 0 or a nonerror [HRESULT](#) (as specified [\[MS-ERREF\]](#)) on success, or an implementation-specific nonzero error code on failure (see section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Unless otherwise specified in this specification, client implementations of the protocol MUST NOT take any action on an error code, but rather simply return the error to the invoking application. If the return code is not an error, the client SHOULD assume that all output parameters are present and valid.<122>

Methods in RPC Opnum Order

Method	Description
IVolumeClient4::RefreshEx	Opnum: 3
IVolumeClient4::GetVolumeDeviceName	Opnum: 4

3.2.4.4.4.1 IVolumeClient4::RefreshEx (Opnum 3)

The **RefreshEx** method refreshes the server's cache of storage objects, including regions, removable media and CD-ROM drive media, file systems, and drive letters.

```
HRESULT RefreshEx();
```

This method has no parameters.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

The handling of this message is identical to the handling of [IVolumeClient::Refresh](#) and [IVolumeClient3::Refresh](#) except that the server MUST perform an extra low-level refresh of the list of storage objects by looking for missing dynamic disks or dynamic disks that were missing and are now present. This verification updates the status for missing disks, volumes that reside on missing disks, or disk regions that reside on missing disks.<123>

In addition to the above, the server MUST check if the length of the disks have changed and make appropriate changes to the disk objects in the list of storage objects.

3.2.4.4.4.2 IVolumeClient4::GetVolumeDeviceName (Opnum 4)

The **GetVolumeDeviceName** method retrieves the Windows NT device name of a dynamic volume on the server.

```
HRESULT GetVolumeDeviceName(
    [in] LdmObjectId _volumeId,
    [out] unsigned long* cchVolumeDevice,
    [out, size_is(*cchVolumeDevice)]
    WCHAR** pwszVolumeDevice
);
```

_volumeId: Specifies the object identifier of the volume whose path name is being returned.

cchVolumeDevice: Number of characters returned in *pwszVolumeDevice*, including null termination.

pwszVolumeDevice: Pointer to a null-terminated array of characters that stores the Windows NT device name of the volume specified by *volumeId*. The device name is in the format \Device\DeviceName. Memory for the array is allocated by the server and freed by the client.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters.

1. Verify that *cchVolumeDevice* and *pwszVolumeDevice* are not NULL.
2. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects.

If parameter validation fails, the server MUST fail the operation immediately, returning an appropriate error as its response to the client.

Otherwise, the server MUST compose a response to the client as follows:

1. Retrieve the device name of the dynamic volume specified by *volumeId*. The device name is an OS-specific name that can be used to access the device from the kernel.
2. Allocate a buffer large enough to contain the device name, including null termination.
3. Populate the buffer with the device name, including null termination.
4. The buffer MUST be returned to the client in the output parameter *pwszVolumeDevice*.
5. The number of characters in the buffer, including null termination, MUST be returned in the output parameter *cchVolumeDevice*.
6. Return a response that contains the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of storage objects as part of processing this message.

3.2.4.4.5 IDMRemoteServer

Methods in RPC Opnum Order

Method	Description
IDMRemoteServer::CreateRemoteObject	Opnum: 3

3.2.4.4.5.1 IDMRemoteServer::CreateRemoteObject (Opnum 3)

The **CreateRemoteObject** method creates a disk management server on the remote machine specified by *RemoteComputerName* by invoking DCOM with the class GUID of Disk Management server and name of the remote machine, which starts the disk management server on the remote machine. The method negotiates for the interface as described in section [3.1.3](#), and as illustrated in section [4](#). The client holds a reference to the [IDMRemoteServer](#) interface binding on the server, until the client has received an [IVolumeClient](#), or [IVolumeClient3](#) interface binding to the remote server. The client MAY then release the **IDMRemoteServer** interface on the server.

```
HRESULT CreateRemoteObject (
```

```

[in] unsigned long cMax,
[in, max_is(cMax)] wchar_t* RemoteComputerName
);

```

cMax: Length of *RemoteComputerName* (in Unicode characters), including null termination.

RemoteComputerName: Null-terminated Unicode string that specifies the name of the computer on which the server is to be activated. All UNC names ("\\server" or "server") and DNS names ("domain.com", "example.microsoft.com", or "135.5.33.19") names are allowed.

Return Values: The method MUST return 0 or a nonerror [HRESULT](#) on success, or an implementation-specific nonzero error code on failure (as specified in [\[MS-ERREF\]](#); see also section [2.2.1](#) for **HRESULT** values predefined by the Disk Management Remote Protocol).

3.2.5 Timer Events

No timers are used by the Disk Management Remote Protocol.

3.2.6 Other Local Events

The server SHOULD register for receiving notifications from the operating system related to changes in the storage configuration of the system. Such changes may be due to administrative change of the hardware configuration, hardware failures, or administrative configuration of storage objects using various tools, for example.

Note Failure to register for and handle storage change notifications impairs the capability of clients to perform configuration operations in the situations enumerated above. [<124>](#)

3.2.6.1 Disk Arrival

When the operating system notifies the server of a new disk connected to the system, the server MUST add a new object to the list of storage objects. A unique identifier MUST be associated with the new object. The **LastKnownState** of the object MUST be initialized with a value at the server's discretion. The server MUST then send a notification of type DMNOTIFY_DISK_INFO with action LDMACTION_CREATED to all clients currently registered for notifications.

Then the server MUST enumerate all the regions that reside on the disk, reevaluate the status of the volumes that use those regions (together with their drive letters and file systems), make appropriate changes to the list of storage objects, and send notifications to all clients currently registered for notifications.

3.2.6.2 Disk Removal

When the operating system notifies the server that a disk has been disconnected from the system, the server MUST remove the disk from the list of storage objects. Then the server MUST send a notification of type DMNOTIFY_DISK_INFO with action LDMACTION_DELETED to all clients currently registered for notifications.

Before taking the preceding steps, the server MUST enumerate the regions that reside on the disk, reevaluate the status of the volumes that use those regions (together with their drive letters and file systems), make appropriate changes to the list of storage objects, and send notifications to all clients currently registered for notifications.

3.2.6.3 Disk Layout Change

When the operating system notifies the server that the partitioning layout of a disk was changed by an entity other than the server, the server MUST re-enumerate the regions that reside on the disk, reevaluate the status of the volumes that use those regions (together with their drive letters and file systems), make appropriate changes to the list of storage objects, and send notifications to all clients currently registered for notifications.

3.2.6.4 File System Change

When the operating system notifies the server that one of the volumes has been formatted by an entity other than the server, the server MUST add a new file system object to the list of storage objects, and then send a notification of type DMNOTIFY_FS_INFO with action LDMACTION_CREATED to all clients currently registered for notifications. The old file system (if any) MUST be removed from the list of storage objects, and the appropriate notifications MUST be sent to all clients currently registered for notifications.

When the operating system notifies the server that the attributes of a file system (for example, the **volume label**) have been changed by an entity other than the server, the server MUST send a notification of type DMNOTIFY_FS_INFO with action LDMACTION_MODIFIED to all clients currently registered for notifications.

3.2.6.5 Drive Letter Arrival

When the operating system notifies the server that one of the volumes has been assigned a drive letter by an entity other than the server, the server MUST send a notification of type DMNOTIFY_DL_INFO with action LDMACTION_MODIFIED to all clients currently registered for notifications.

3.2.6.6 Drive Letter Removal

When the operating system notifies the server that a drive letter has been removed by an entity other than the server, the server MUST send a notification of type DMNOTIFY_DL_INFO with action LDMACTION_MODIFIED to all clients currently registered for notifications.

3.2.6.7 Media Arrival

When the operating system notifies the server that media has been inserted by the user in a removable, CD-ROM, or DVD unit, the server MUST send a notification of type DMNOTIFY_DISK_INFO with action LDMACTION_MODIFIED to all clients currently registered for notifications.

Then the server MUST enumerate all the regions that reside on the disk (together with their drive letters and file systems), make appropriate changes to the list of storage objects, and send notifications to all clients currently registered for notifications.

3.2.6.8 Media Removal

When the operating system notifies the server that media has been removed by the user from a removable disk, CD-ROM, or DVD unit, the server MUST send a notification of type DMNOTIFY_DISK_INFO with action LDMACTION_MODIFIED to all clients currently registered for notifications.

Before doing that, the server MUST enumerate all the regions that reside on the disk (together with their drive letters and file systems), make appropriate changes to the list of storage objects, and send notifications to all clients currently registered for notifications.

4 Protocol Examples

4.1 Starting a New Session on a Local or Remote Server

The following diagram shows how the client would start a new session on a local or remote server.

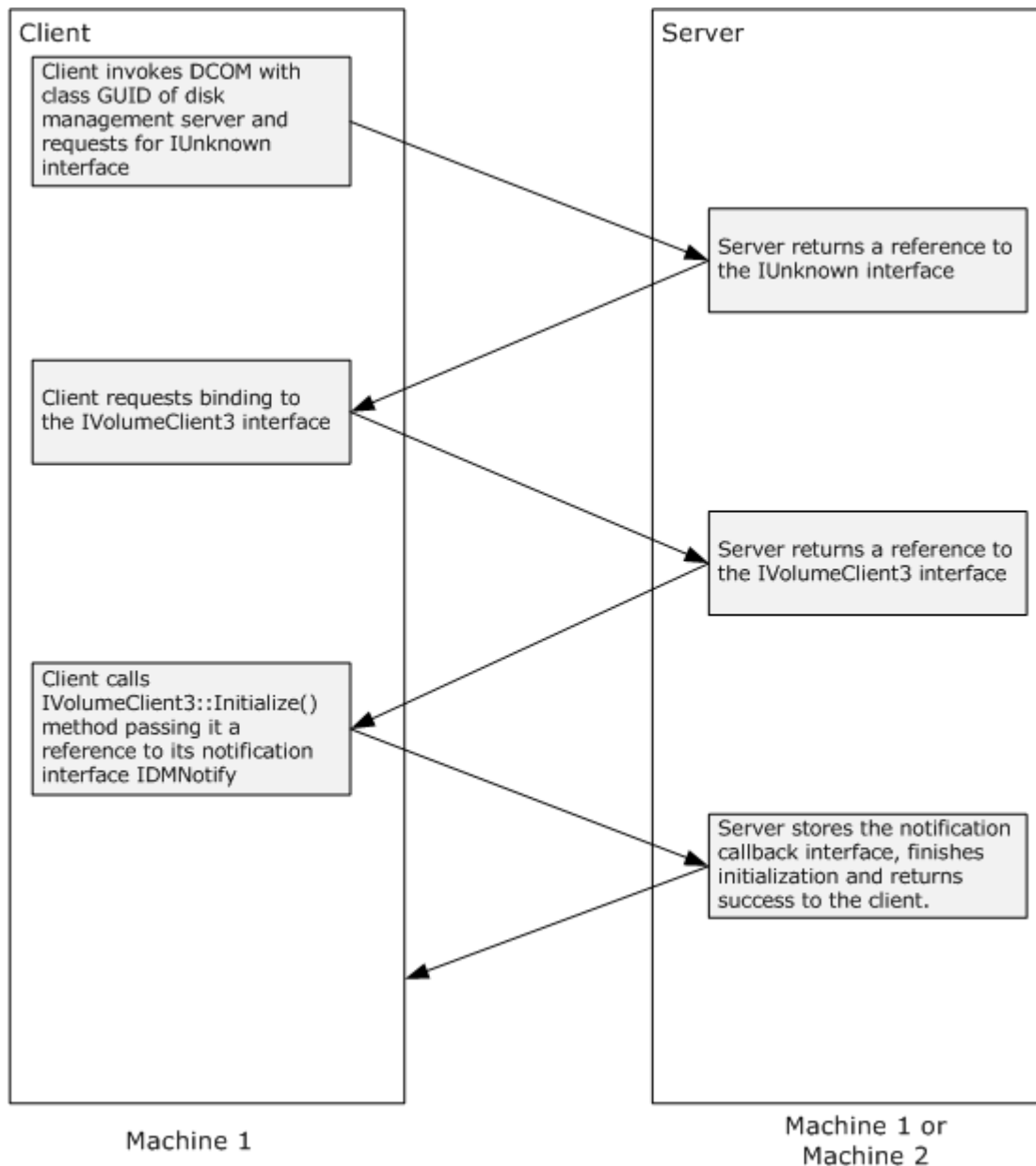


Figure 1: Steps to start a new session on a local or remote server

1. The client starts a new session of the disk management server by invoking DCOM with the class GUID of the disk management server and requests for **IUnknown** interface.

2. The server returns a reference to the **IUnknown** interface.
3. The client requests binding to the [IVolumeClient3](#) interface.
4. The server returns a reference to the **IVolumeClient3** interface.
5. The client calls the [IVolumeClient3::Initialize\(\)](#) method passing it a reference to its notification callback interface [IDMNotify](#).
6. The server stores the notification callback interface, finishes the initialization and returns success to the client.

4.2 Starting a New Session on a Remote Server Using the IDMRemoteServer Interface

The following diagram shows how the client would start a new session on remote server using [IDMRemoteServer](#) interface.

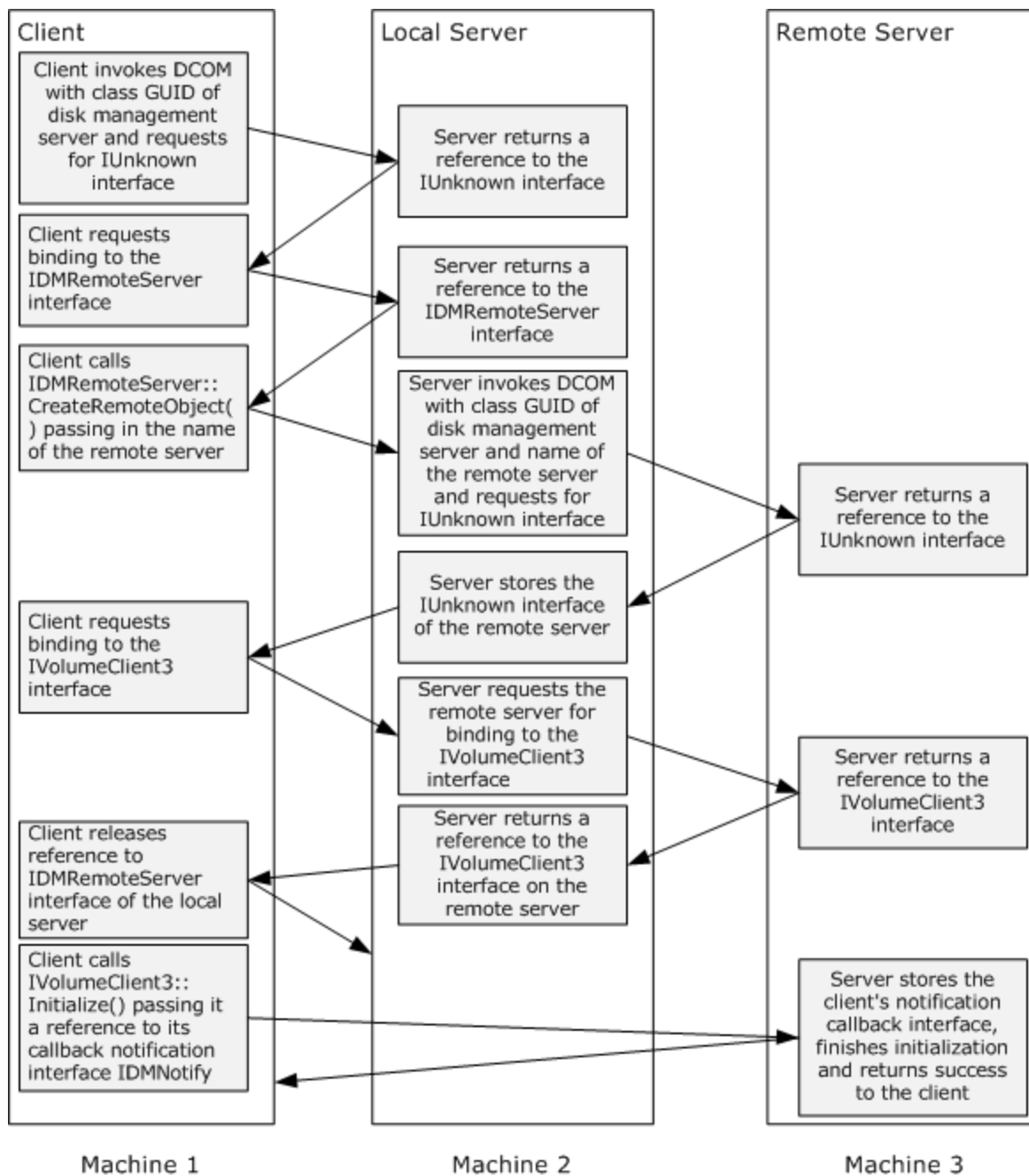


Figure 2: Steps to start a new session on a remote server using IDmRemoteServer interface

1. The client starts a new session of the local disk management server by invoking DCOM with the class GUID of the disk management server and requests for **IUnknown** interface.
2. The local server returns a reference to the **IUnknown** interface.
3. The client requests binding to the **IDmRemoteServer** interface.

4. The local server returns a reference to the **IDmRemoteServer** interface.
5. The client calls [IDMRemoteServer::CreateRemoteObject \(Opnum 3\)](#) method passing it the name of the remote server.
6. The local server does the following:
 - Starts the disk management server on remote server by invoking DCOM with the class GUID of the disk management server and remote server name, and, requests the **IUnknown** interface.
 - Receives an **IUnknown** interface pointer to the remote server.
 - Stores the reference to the **IUnknown** interface of the remote server.
7. The client requests binding to the [IVolumeClient3](#) interface.
8. The local server in turn calls into the remote server using the stored **IUnknown** interface pointer and requests binding to the **IVolumeClient3** interface. The local server returns a reference to the **IVolumeClient3** interface on the remote server to the client.
9. The client may now release the **IDmRemoteServer** interface, since it now holds an interface pointer to the remote server.
10. The client calls the [IVolumeClient3::Initialize\(\)](#) method passing it a reference to its notification callback interface [IDMNotify](#).
11. The remote server stores the client's **IDMNotify** notification callback, finishes the initialization, and returns success to the client.

4.3 Creating a Partition

The following diagram shows how the [IVolumeClient](#) interfaces are used to create a partition on a disk.

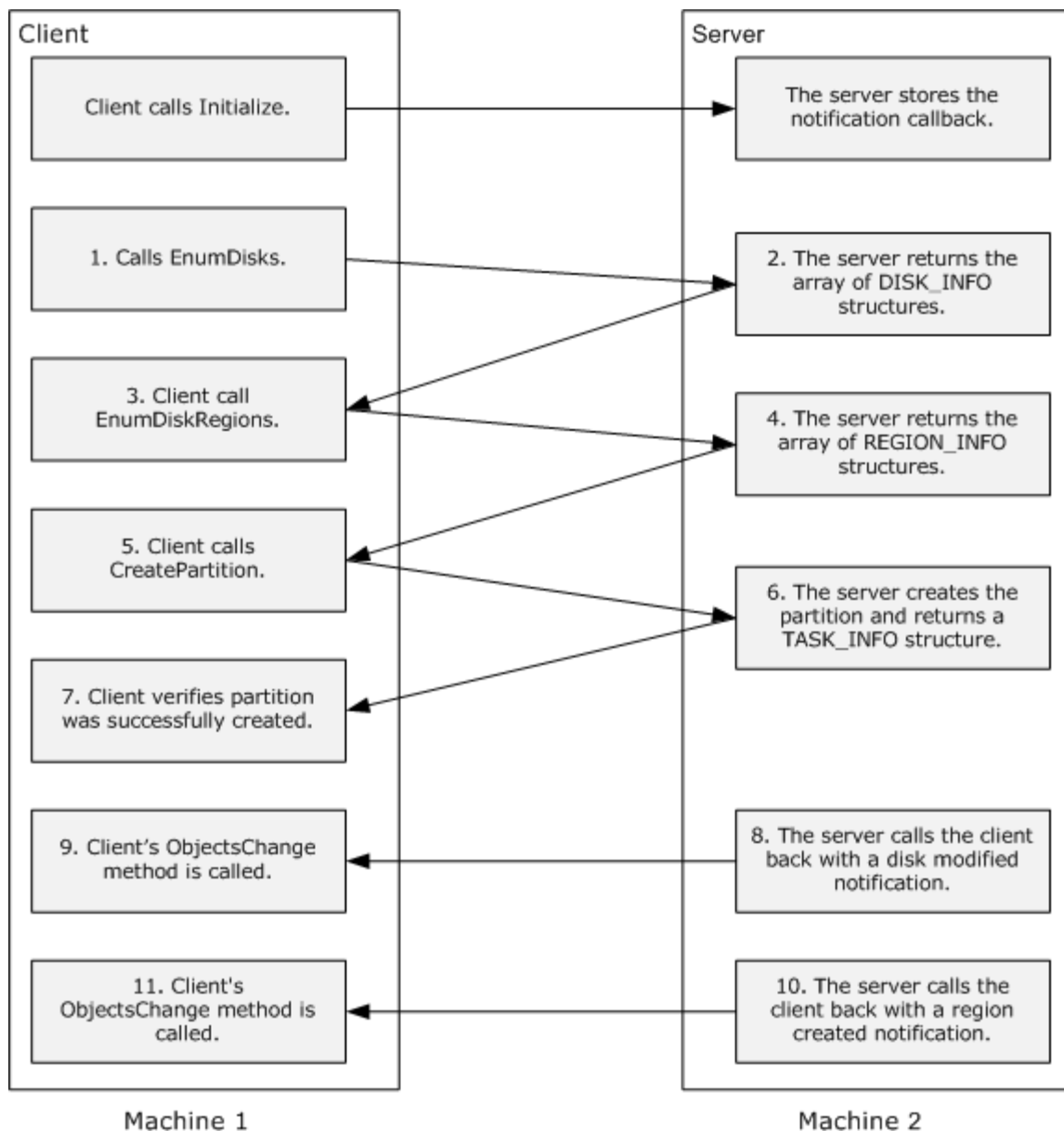


Figure 3: Steps to create a partition on a disk by using the IVolumeClient interfaces

1. The client calls [IVolumeClient::Initialize](#) and passes its implementation of the [IDMNotify](#) interface to the server.
2. The client calls [IVolumeClient::EnumDisks](#) to get the list of disks from the remote machine.
3. For each disk, a [DISK_INFO](#) structure is returned. The server allocates the memory and returns the array of [DISK_INFO](#) structures and [HRESULT](#) to the client.
4. The client verifies that the call was successful by looking at the returned [HRESULT](#). If the call was successful, the client finds the disk to be used to create the partition (for example, by looking at the disk name field in the [DISK_INFO](#) structure). The client then calls

[IVolumeClient::EnumDiskRegions](#) to get an array that represents the regions on the disk. The input parameter to **EnumDiskRegions** is the disk's [LdmObjectId](#), which is the first member of the **DISK_INFO** structure.

5. The server allocates an array of [REGION_INFO](#) structures and returns the array to the client.
6. The client verifies that the call was successful by looking at the returned **HRESULT**. If the call was successful, the client parses the array to find a region with **REGIONTYPE REGION_FREE**. The client looks at the size of the region to see what size partition can be created in this region. If this free region is not large enough for the expected partition size, the client can continue to look for free regions on that disk that are larger. Once the client finds the region to be used to create the new volume, the client calls [IVolumeClient::CreatePartition](#). The input parameter to the call is a [REGION_SPEC](#) structure. The client fills in the **LdmObjectId** for the region - this information is obtained from the **REGION_INFO** structure. The region type would be **REGION_PRIMARY** to create a primary partition. The client fills in the disk id and *lastKnownState* members of the **REGION_SPEC** structure by using the values from the **REGION_INFO** structure. The client also fills in the start and length fields of the **REGION_SPEC**. The length must be no greater than the reported region length from the **REGION_INFO** structure. The start must be within the offsets that represent the start and end of the region.
7. The server creates the partition and fills in the [TASK_INFO](#) structure. The call to **IVolumeClient::CreatePartition** returns this **TASK_INFO** structure.
8. The client verifies that the partition was successfully created by looking at the **HRESULT** returned from the call. The returned **TASK_INFO** structure will contain the new region's id in the **storageId** field. The **status** field in the **TASK_INFO** structure will be **REQ_COMPLETED**.
9. The server calls back the client with a disk modified notification on the **IDMNotify** interface's [ObjectsChanged](#) method.
10. The client code processes the disk modified notification. For example, the client may query for the current disk information and all diskregions when it gets a disk modified notification, so that it can update its cache or display.
11. The server calls back the client with a region created notification on the **IDMNotify** interface's **ObjectsChanged** method.
12. The client code processes the region created notification. For example, the client may query for all diskregions when it gets a region created notification, so that it can update its cache or display.

4.4 Deleting a Partition

The following diagram shows how the [IVolumeClient](#) interfaces are used to delete a partition on a disk.

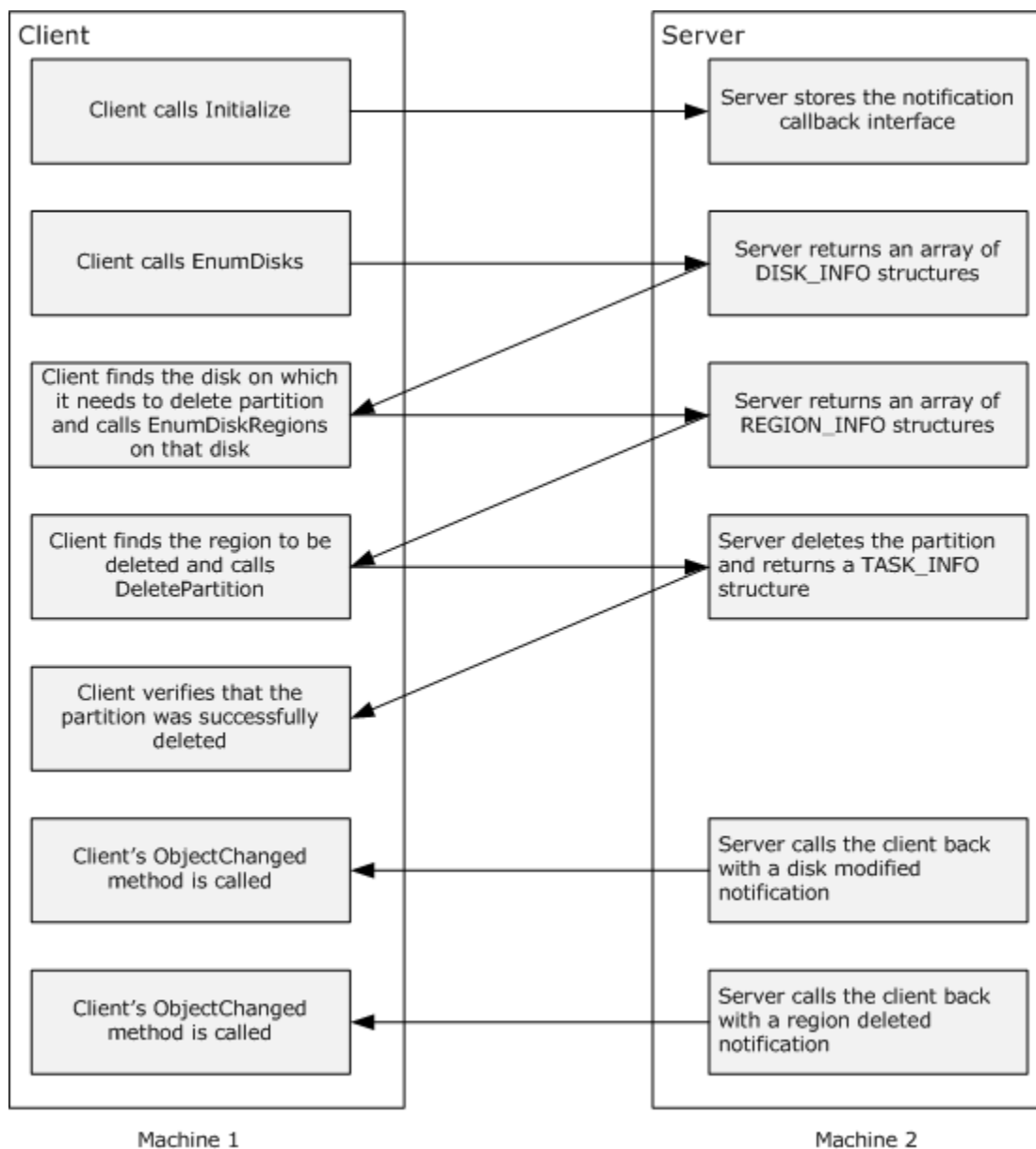


Figure 4: Steps to delete a partition on a disk by using the IVolumeClient interfaces

1. The client calls [IVolumeClient::Initialize](#) and passes its implementation of the [IDMNotify](#) interface to the server.
2. The client calls [IVolumeClient::EnumDisks](#) to get the list of disks from the server.
3. For each disk, a [DISK_INFO](#) structure is returned. The server allocates the memory and returns the array of **DISK_INFO** structures and HRESULT to the client.
4. The client verifies that the call was successful by looking at the returned HRESULT. If the call was successful, the client finds the disks to be used to delete the partition (for example, by looking at the disk name field in the **DISK_INFO** structure). The client then calls

[IVolumeClient::EnumDiskRegions](#) to get an array that represents the regions on the disk on which it wants to delete the partition. The input parameter to **EnumDiskRegions** is the disk's [LdmObjectId](#), which is the first member of the **DISK_INFO** structure.

5. The server allocates an array of [REGION_INFO](#) structures and returns the array to the client.
6. The client verifies that the call was successful by looking at the returned HRESULT. If the call was successful, the client parses the array to find a region which it wants to delete. Once the client finds the region to be deleted, the client calls [IVolumeClient::DeletePartition](#). The input parameter to the call is a [REGION_SPEC](#) structure and bForce flag. The client fills in the **LdmObjectId** for the region - this information is obtained from the **REGION_INFO** structure. The client fills in the region type, disk id and lastKnownState members of the **REGION_SPEC** structure by using the values from the **REGION_INFO** structure. The client also fills in the **start** and **length** fields of the **REGION_SPEC**. The client sets the *force* parameter to TRUE if it wants to force the deletion of the partition; otherwise, it sets the *force* parameter to FALSE.
7. The server deletes the partition and fills in the [TASK_INFO](#) structure. The call to **IVolumeClient::DeletePartition** returns this **TASK_INFO** structure.
8. The client verifies that the partition was successfully deleted by looking at the HRESULT returned from the call. The returned **TASK_INFO** structure will contain the deleted region's id in the **storageId** field. The **status** field in the **TASK_INFO** structure will be REQ_COMPLETED.
9. The server calls back the client with a disk modified notification on the IDMNofity Interface's [ObjectsChanged](#) method.
10. The client code processes the disk modified notification. For example, the client may query for the current disk information and all disk regions when it gets a disk modified notification, so that it can update its cache or display.
11. The server calls back the client with a region deleted notification on the IDMNofity Interface's **ObjectsChanged** method.
12. The client code processes the region deleted notification. For example, the client may query for all diskregions when it gets a region deleted notification, so that it can update its cache or display.

4.5 Creating a Volume

The following diagram shows how the [IVolumeClient](#) interfaces are used to create a volume on a disk.

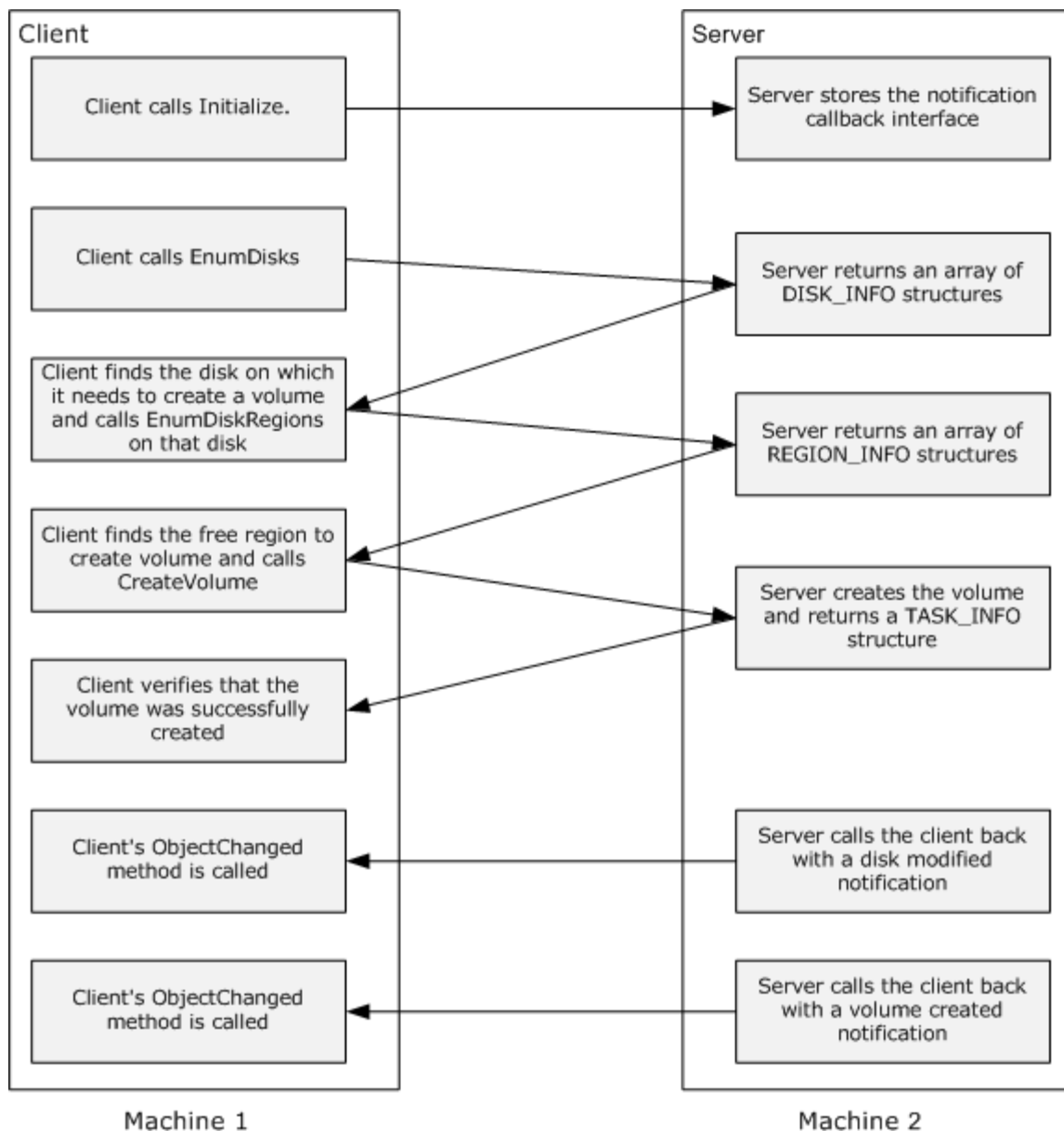


Figure 5: Steps to create a volume on a disk by using the IVolumeClient interfaces

1. The client calls [IVolumeClient::Initialize](#) and passes its implementation of the [IDMNotify](#) interface to the server.
2. The client calls [IVolumeClient::EnumDisks](#) to get the list of disks from the server.
3. For each disk, a [DISK_INFO](#) structure is returned. The server allocates the memory and returns the array of **DISK_INFO** structures and HRESULT to the client.
4. The client verifies that the call was successful by looking at the returned HRESULT. If the call was successful, the client finds the disks to be used to create the volume (for example, by looking at the disk name field in the **DISK_INFO** structure). The client then calls

[IVolumeClient::EnumDiskRegions](#) to get an array that represents the regions on the disk. The input parameter to **EnumDiskRegions** is the disk's [LdmObjectId](#), which is the first member of the **DISK_INFO** structure.

5. The server allocates an array of [REGION_INFO](#) structures and returns the array to the client.
6. The client verifies that the call was successful by looking at the returned HRESULT. If the call was successful, the client parses the array to find a region with REGIONTYPE REGION_FREE. The client looks at the size of the region to see what size volume can be created in this region. If this free region is not large enough for the expected volume size, the client can continue to look for larger free regions on that disk. The client can repeat the calls to **IVolumeClient::EnumDiskRegions** on all the disks until it finds the set of disks with free space required to create the volume of expected size. Once the client finds the region to be used to create the new volume, the client calls [IVolumeClient::CreateVolume](#). The input parameter to the call is an array of [DISK_SPEC](#) structures and [VOLUME_SPEC](#) structures. The client fills in the **diskid** and **lastKnownState** members of the **DISK_SPEC** structure by using the values from the **DISK_INFO** structure. The client fills in the **length** field based on the size of the volume to be created on the particular disk. The length must be no greater than the reported free region length from the **REGION_INFO** structure. The client also fills in the **needContiguous** field based on the type of the volume. The client fills in the **VOLUME_SPEC** structure based on the volume type and size of the volume to be created.
7. The server creates the volume and fills in the [TASK_INFO](#) structure. The call to **IVolumeClient::CreateVolume** returns this **TASK_INFO** structure.
8. The client verifies that the volume was successfully created by looking at the HRESULT returned from the call. The returned **TASK_INFO** structure will contain the new volume's id in the **storageId** field. The **status** field in the **TASK_INFO** structure will be REQ_COMPLETED.
9. The server calls back the client with a disk modified notification on the **IDMNotify** interface's [ObjectsChanged](#) method.
10. The client code processes the disk modified notification. For example, the client may query for the current disk information and all disk regions when it gets a disk modified notification, so that it can update its cache or display.
11. The server calls back the client with a volume created notification on the **IDMNotify** interface's **ObjectsChanged** method.
12. The client code processes the volume created notification. For example, the client may query for all volumes when it gets a volume created notification, so that it can update its cache or display.

4.6 Deleting a Volume

The following diagram shows how the [IVolumeClient](#) interfaces are used to delete a volume.

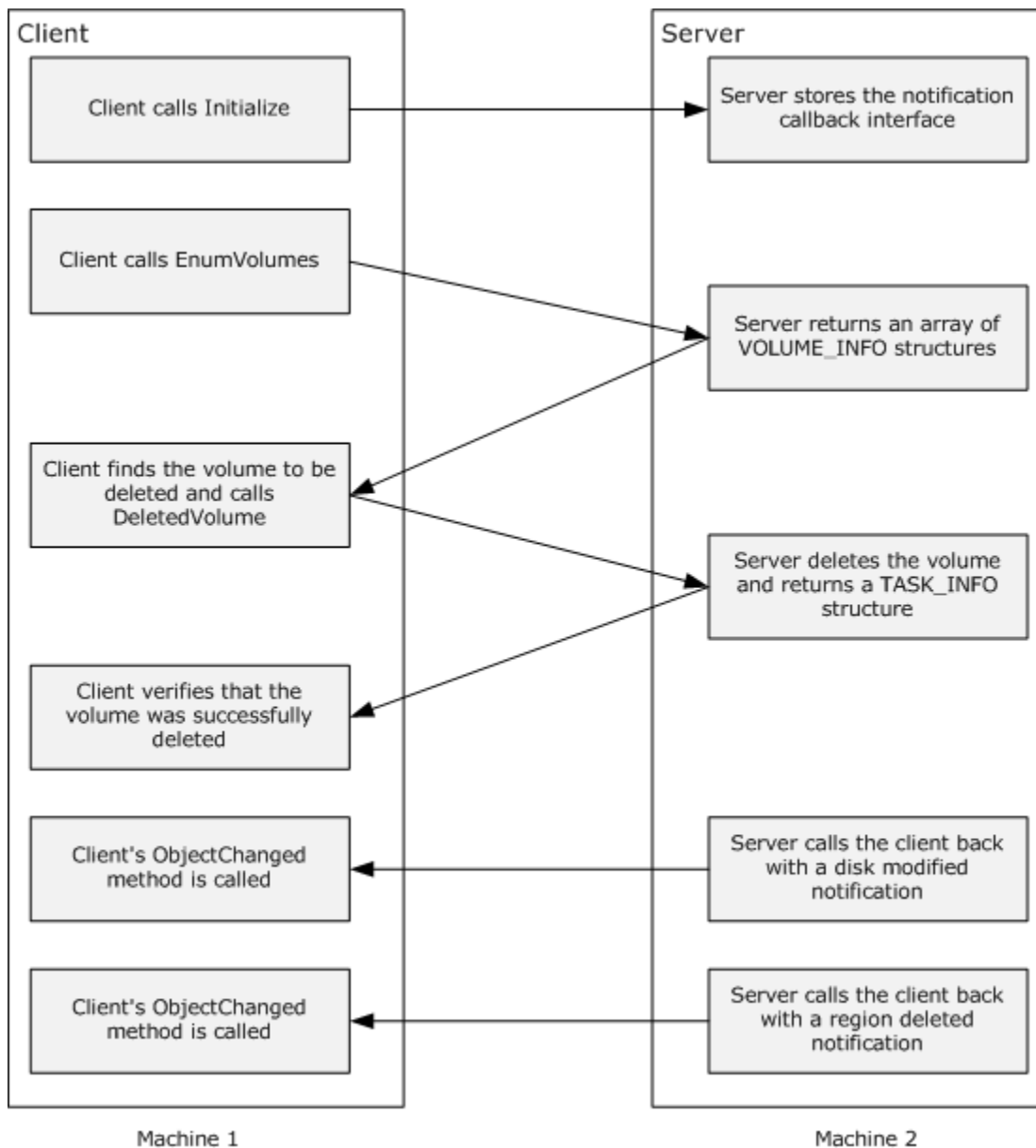


Figure 6: Steps to delete a volume on a disk by using the IVolumeClient interfaces

1. The client calls [IVolumeClient::Initialize](#) and passes its implementation of the [IDMNotify Interface](#) to the server.
2. The client calls [IVolumeClient::EnumVolumes](#) to get the list of volumes from the server.
3. For each volume, a [VOLUME_INFO](#) structure is returned. The server allocates the memory and returns the array of **VOLUME_INFO** structures and HRESULT to the client.
4. The client verifies that the call was successful by looking at the returned HRESULT. If the call was successful, the client finds the volume to be deleted (for example, by looking at the **id** field in the **VOLUME_INFO** structure). Once the client finds the volume to be deleted, it calls

[IVolumeClient::DeleteVolume](#) to delete the volume. The input parameter to [DeleteVolume](#) is the volume's [LdmObjectId](#), which is the first member of the **VOLUME_INFO** structure. The client sets the *force* parameter to TRUE if it wants to force the deletion of the partition; otherwise, it sets the *force* parameter to FALSE. The client sets the *volumeLastKnownState* parameter from the **VOLUME_INFO** structure.

5. The server deletes the volume and fills in the [TASK_INFO](#) structure. The call to **IVolumeClient::DeleteVolume** returns this **TASK_INFO** structure.
6. The client verifies that the volume was successfully deleted by looking at the HRESULT returned from the call. The returned **TASK_INFO** structure will contain the deleted volume's id in the **storageId** field. The **status** field in the **TASK_INFO** structure will be REQ_COMPLETED.
7. The server calls back the client with a disk modified notification on the **IDMNotify** interface's [ObjectsChanged](#) method.
8. The client code processes the disk modified notification. For example, the client may query for the current disk information and all disk regions when it gets a disk modified notification, so that it can update its cache or display.
9. The server calls back the client with a volume deleted notification on the **IDMNotify** interface's **ObjectsChanged** method.
10. The client code processes the volume deleted notification. For example, the client may query for all volumes when it gets a volume deleted notification, so that it can update its cache or display.

5 Security Considerations

This protocol introduces no security considerations beyond those applicable to DCOM interfaces, as specified in [\[MS-DCOM\]](#) section 5.

6 Appendix A: Full IDL

6.1 dmintf.idl

Information about this .idl must be obtained from Veritas Software Corporation.

6.2 dmintf3.idl

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

const DWORD SYSFLAG_NO_DYNAMIC = 0x10;
const DWORD SYSFLAG_IA64 = 0x40;
const DWORD SYSFLAG_UNINSTALL_VALID = 0x80;
const DWORD SYSFLAG_DYNAMIC_1394 = 0x100;

typedef enum _PARTITIONSTYLE {
    PARTITIONSTYLE_UNKNOWN = 0,
    PARTITIONSTYLE_MBR,
    PARTITIONSTYLE_GPT
} PARTITIONSTYLE;

struct diskinfoex {
    LdmObjectId id;
    LONGLONG length;
    LONGLONG freeBytes;
    unsigned long bytesPerTrack;
    unsigned long bytesPerCylinder;
    unsigned long bytesPerSector;
    unsigned long regionCount;
    unsigned long dflags;
    unsigned long deviceType;
    unsigned long deviceState;
    unsigned long busType;
    unsigned long attributes;
    unsigned long maxPartitionCount;
    boolean isUpgradeable;
    boolean maySwitchStyle;
    PARTITIONSTYLE partitionStyle;
    [switch_is(partitionStyle)] union {
        [case(PARTITIONSTYLE_MBR)] struct {
            unsigned long signature;
        } mbr;
        [case(PARTITIONSTYLE_GPT)] struct {
            GUID diskId;
        } gpt;
    } [default];
};

int portNumber;
int targetNumber;
int lunNumber;
LONGLONG lastKnownState;
LdmObjectId taskId;
int cchName;
```

```

    int                cchVendor;
    int                cchDgid;
    int                cchAdapterName;
    int                cchDgName;
    int                cchDevInstId;
    [size_is(cchName)] wchar_t * name;
    [size_is(cchVendor)] wchar_t * vendor;
    [size_is(cchDgid)] byte * dgid;
    [size_is(cchAdapterName)] wchar_t * adapterName;
    [size_is(cchDgName)] wchar_t * dgName;
    [size_is(cchDevInstId)] wchar_t * devInstId;
};

typedef struct diskinfoex DISK_INFO_EX;

const DWORD DISK_FORMATTABLE_DVD      = 0x4;
const DWORD DISK_MEMORY_STICK         = 0x8;
const DWORD DISK_NTFS_NOT_SUPPORTED = 0x10;

struct regioninfoex {
    LdmObjectId        id;
    LdmObjectId        diskId;
    LdmObjectId        volId;
    LdmObjectId        fsId;
    LONGLONG           start;
    LONGLONG           length;
    REGIONTYPE         regionType;
    PARTITIONSTYLE     partitionStyle;
    [switch_is(partitionStyle)] union {
        [case(PARTITIONSTYLE_MBR)] struct {
            unsigned long  partitionType;
            boolean        isActive;
        } mbr;
        [case(PARTITIONSTYLE_GPT)] struct {
            GUID           partitionType;
            GUID           partitionId;
            ULONGLONG      attributes;
        } gpt;
    } [default];
};

REGIONSTATUS          status;
hyper                 lastKnownState;
LdmObjectId           taskId;
unsigned long          rflags;
unsigned long          currentPartitionNumber;
int                   cchName;
[size_is(cchName)] wchar_t * name;
};

typedef struct regioninfoex REGION_INFO_EX;

const DWORD REGION_HIDDEN= 0x40000;

const DWORD ENCAP_INFO_MIXED_PARTITIONS = 0x1000;
const DWORD ENCAP_INFO_OPEN_FAILED     = 0x2000;

[ object, uuid(135698D2-3A37-4d26-99DF-E2BB6AE3AC61),
  pointer_default(unique) ]
interface IVolumeClient3 : IUnknown

```

```

{
    HRESULT EnumDisksEx([out] unsigned long *diskCount,
                        [out, size_is(*diskCount)] DISK_INFO_EX
**diskList);

    HRESULT EnumDiskRegionsEx([in] LdmObjectId diskId,
                              [in, out] unsigned long *numRegions,
                              [out, size_is(*numRegions)]
REGION_INFO_EX **regionList);

    HRESULT CreatePartition([in] REGION_SPEC partitionSpec,
                            [out] TASK_INFO *tinfo);

    HRESULT CreatePartitionAssignAndFormat([in] REGION_SPEC
partitionSpec,
                                           [in] wchar_t letter,
                                           [in] hyper letterLastKnownState,
                                           [in] FILE_SYSTEM_INFO fsSpec,
                                           [in] boolean quickFormat,
                                           [out] TASK_INFO *tinfo);

    HRESULT CreatePartitionAssignAndFormatEx([in] REGION_SPEC
partitionSpec,
                                           [in] wchar_t letter,
                                           [in] hyper letterLastKnownState,
                                           [in] int cchAccessPath,
                                           [in, size_is(cchAccessPath)] wchar_t
*AccessPath,
                                           [in] FILE_SYSTEM_INFO fsSpec,
                                           [in] boolean quickFormat,
                                           [in] DWORD dwFlags,
                                           [out] TASK_INFO *tinfo);

    HRESULT DeletePartition([in] REGION_SPEC partitionSpec,
                            [in] boolean force,
                            [out] TASK_INFO *tinfo);

    HRESULT InitializeDiskStyle([in] LdmObjectId diskId,
                                [in] PARTITIONSTYLE style,
                                [in] hyper diskLastKnownState,
                                [out] TASK_INFO *tinfo);

    HRESULT MarkActivePartition([in] LdmObjectId regionId,
                                [in] hyper regionLastKnownState,
                                [out] TASK_INFO *tinfo );

    HRESULT Eject( [in] LdmObjectId diskId,
                   [in] hyper diskLastKnownState,
                   [out] TASK_INFO *tinfo );

    HRESULT Reserved_Opnum12(void);

    HRESULT FTEnumVolumes([in, out] unsigned long *volumeCount,
                          [out, size_is(*volumeCount)] VOLUME_INFO **ftVolumeList);

    HRESULT FTEnumLogicalDiskMembers([in] LdmObjectId volumeId,
                                      [in, out] unsigned long *memberCount,
                                      [out, size_is(*memberCount)] LdmObjectId **memberList);

```

```

HRESULT FTDeleteVolume([in] LdmObjectId volumeId,
                       [in] boolean force,
                       [in] hyper volumeLastKnownState,
                       [out] TASK_INFO *tinfo);

HRESULT FTBreakMirror([in] LdmObjectId volumeId,
                     [in] hyper volumeLastKnownState,
                     [in] boolean bForce,
                     [out] TASK_INFO *tinfo);

HRESULT FTResyncMirror([in] LdmObjectId volumeId,
                      [in] hyper volumeLastKnownState,
                      [out] TASK_INFO *tinfo);

HRESULT FTRegenerateParityStripe([in] LdmObjectId volumeId,
                                 [in] hyper volumeLastKnownState,
                                 [out] TASK_INFO *tinfo);

HRESULT FTReplaceMirrorPartition([in] LdmObjectId volumeId,
                                 [in] hyper volumeLastKnownState,
                                 [in] LdmObjectId oldMemberId,
                                 [in] hyper oldMemberLastKnownState,
                                 [in] LdmObjectId newRegionId,
                                 [in] hyper newRegionLastKnownState,
                                 [in] DWORD flags,
                                 [out] TASK_INFO *tinfo);

HRESULT FTReplaceParityStripePartition([in] LdmObjectId volumeId,
                                       [in] hyper volumeLastKnownState,
                                       [in] LdmObjectId oldMemberId,
                                       [in] hyper oldMemberLastKnownState,
                                       [in] LdmObjectId newRegionId,
                                       [in] hyper newRegionLastKnownState,
                                       [in] DWORD flags,
                                       [out] TASK_INFO *tinfo);

HRESULT EnumDriveLetters([in, out] unsigned long *
driveLetterCount,
                        [out, size_is(*driveLetterCount)] DRIVE_LETTER_INFO
**driveLetterList);

HRESULT AssignDriveLetter([in] wchar_t letter,
                         [in] unsigned long forceOption,
                         [in] hyper letterLastKnownState,
                         [in] LdmObjectId storageId,
                         [in] hyper storageLastKnownState,
                         [out] TASK_INFO *tinfo);

HRESULT FreeDriveLetter([in] wchar_t letter,
                       [in] unsigned long forceOption,
                       [in] hyper letterLastKnownState,
                       [in] LdmObjectId storageId,
                       [in] hyper storageLastKnownState,
                       [out] TASK_INFO *tinfo);

HRESULT EnumLocalFileSystems([out] unsigned long * fileSystemCount,
                             [out, size_is(*fileSystemCount)] FILE_SYSTEM_INFO

```

```

**fileSystemList);

HRESULT GetInstalledFileSystems([out] unsigned long *fsCount,
    [out, size_is(*fsCount)] IFILE_SYSTEM_INFO **fsList);

HRESULT Format([in] LdmObjectId storageId,
    [in] FILE_SYSTEM_INFO fsSpec,
    [in] boolean quickFormat,
    [in] boolean force,
    [in] hyper storageLastKnownState,
    [out] TASK_INFO *tinfo);

HRESULT EnumVolumes(
    [in, out] unsigned long *volumeCount,
    [out, size_is(*volumeCount)] VOLUME_INFO **LdmVolumeList);

HRESULT EnumVolumeMembers([in] LdmObjectId volumeId,
    [in, out] unsigned long * memberCount,
    [out, size_is(*memberCount)] LdmObjectId ** memberList);

HRESULT CreateVolume([in] VOLUME_SPEC volumeSpec,
    [in] unsigned long diskCount,
    [in, size_is(diskCount)] DISK_SPEC *diskList,
    [out] TASK_INFO *tinfo );

HRESULT CreateVolumeAssignAndFormat([in] VOLUME_SPEC volumeSpec,
    [in] unsigned long diskCount,
    [in, size_is(diskCount)] DISK_SPEC *diskList,
    [in] wchar_t letter,
    [in] hyper letterLastKnownState,
    [in] FILE_SYSTEM_INFO fsSpec,
    [in] boolean quickFormat,
    [out] TASK_INFO *tinfo);

HRESULT CreateVolumeAssignAndFormatEx([in] VOLUME_SPEC volumeSpec,
    [in] unsigned long diskCount,
    [in, size_is(diskCount)] DISK_SPEC *diskList,
    [in] wchar_t letter,
    [in] hyper letterLastKnownState,
    [in] int cchAccessPath,
    [in, size_is(cchAccessPath)] wchar_t
*AccessPath,
    [in] FILE_SYSTEM_INFO fsSpec,
    [in] boolean quickFormat,
    [in] DWORD dwFlags,
    [out] TASK_INFO *tinfo);

HRESULT GetVolumeMountName( [in] LdmObjectId volumeId,
    [out] unsigned long *cchMountName,
    [out, size_is(*cchMountName)]
WCHAR **mountName);

HRESULT GrowVolume( [in] LdmObjectId volumeId,
    [in] VOLUME_SPEC volumeSpec,
    [in] unsigned long diskCount,
    [in, size_is(diskCount)] DISK_SPEC *diskList,
    [in] boolean force,

```



```

        [out] TASK_INFO *tinfo );

HRESULT DeleteVolume([in] LdmObjectId volumeId,
                    [in] boolean force,
                    [in] hyper volumeLastKnownState,
                    [out] TASK_INFO *tinfo );

HRESULT CreatePartitionsForVolume([in] LdmObjectId volumeId,
                                  [in] boolean active,
                                  [in] hyper volumeLastKnownState,
                                  [out] TASK_INFO *tinfo );

HRESULT DeletePartitionsForVolume([in] LdmObjectId volumeId,
                                  [in] hyper volumeLastKnownState,
                                  [out] TASK_INFO *tinfo );

HRESULT GetMaxAdjustedFreeSpace([in] LdmObjectId diskId,
                                [out] LONGLONG* maxAdjustedFreeSpace);

HRESULT AddMirror([in] LdmObjectId volumeId,
                 [in] hyper volumeLastKnownState,
                 [in] DISK_SPEC diskSpec,
                 [in, out] int *diskNumber,
                 [out] int *partitionNumber,
                 [out] TASK_INFO *tinfo );

HRESULT RemoveMirror([in] LdmObjectId volumeId,
                    [in] hyper volumeLastKnownState,
                    [in] LdmObjectId diskId,
                    [in] hyper diskLastKnownState,
                    [out] TASK_INFO *tinfo );

HRESULT SplitMirror( [in] LdmObjectId volumeId,
                    [in] hyper volumeLastKnownState,
                    [in] LdmObjectId diskId,
                    [in] hyper diskLastKnownState,
                    [in] wchar_t letter,
                    [in] hyper letterLastKnownState,
                    [in, out] TASK_INFO *tinfo );

HRESULT InitializeDiskEx([in] LdmObjectId diskId,
                        [in] PARTITIONSTYLE style,
                        [in] hyper diskLastKnownState,
                        [out] TASK_INFO *tinfo );

HRESULT UninitializeDisk([in] LdmObjectId diskId,
                        [in] hyper diskLastKnownState,
                        [out] TASK_INFO *tinfo );

HRESULT ReConnectDisk( [in] LdmObjectId diskId,
                      [out] TASK_INFO *tinfo );

HRESULT ImportDiskGroup ([in] int cchDgid,
                        [in, size_is( cchDgid)] byte *dgid,
                        [out] TASK_INFO *tinfo);

HRESULT DiskMergeQuery([in] int cchDgid,
[in, size_is( cchDgid)] byte *dgid,

```

```

[in] int numDisks,
[in, size_is( numDisks)] LdmObjectId *diskList,
[out] hyper *merge_config_tid,
[out] int *numRids,
[out, size_is(*numRids)] hyper **merge_dm_rids,
[out] int *numObjects,
[out, size_is(*numObjects)] MERGE_OBJECT_INFO
**mergeObjectInfo,
[in, out] unsigned long *flags,
[out] TASK_INFO *tinfo);

HRESULT DiskMerge([in] int cchDgid,
[in, size_is( cchDgid)] byte *dgid,
[in] int numDisks,
[in, size_is( numDisks)] LdmObjectId *diskList,
[in] hyper merge_config_tid,
[in] int numRids,
[in, size_is(numRids)] hyper *merge_dm_rids,
[out] TASK_INFO *tinfo);

HRESULT ReAttachDisk([in] LdmObjectId diskId,
[in] hyper diskLastKnownState,
[out] TASK_INFO *tinfo );

HRESULT ReplaceRaid5Column([in] LdmObjectId volumeId,
[in] hyper volumeLastKnownState,
[in] LdmObjectId newDiskId,
[in] hyper diskLastKnownState,
[out] TASK_INFO *tinfo );

HRESULT RestartVolume([in] LdmObjectId volumeId,
[in] hyper volumeLastKnownState,
[out] TASK_INFO *tinfo );

HRESULT GetEncapsulateDiskInfoEx( [in] unsigned long diskCount,
[in, size_is(diskCount)] DISK_SPEC *diskSpecList,
[out] unsigned long *encapInfoFlags,
[out] unsigned long *affectedDiskCount,
[out, size_is( ,*affectedDiskCount)] DISK_INFO_EX
**affectedDiskList,
[out, size_is( ,*affectedDiskCount)] unsigned long
**affectedDiskFlags,
[out] unsigned long *affectedVolumeCount,
[out, size_is( ,*affectedVolumeCount)] VOLUME_INFO
**affectedVolumeList,
[out] unsigned long *affectedRegionCount,
[out, size_is( ,*affectedRegionCount)]
REGION_INFO_EX **affectedRegionList,
[out] TASK_INFO *tinfo );

HRESULT EncapsulateDiskEx([in] unsigned long affectedDiskCount,
[in, size_is(affectedDiskCount)] DISK_INFO_EX
*affectedDiskList,
[in] unsigned long affectedVolumeCount,
[in, size_is(affectedVolumeCount)] VOLUME_INFO
*affectedVolumeList,
[in] unsigned long affectedRegionCount,
[in, size_is(affectedRegionCount)] REGION_INFO_EX

```

```

*affectedRegionList,
    [out] unsigned long *encapInfoFlags,
    [out] TASK_INFO *tinfo );

HRESULT QueryChangePartitionNumbers([out] int *oldPartitionNumber,
    [out] int *newPartitionNumber );

HRESULT DeletePartitionNumberInfoFromRegistry();

HRESULT SetDontShow([in] boolean bSetNoShow);

HRESULT GetDontShow([out] boolean *bGetNoShow);

HRESULT Reserved0(
void
);

HRESULT Reserved1(
void
);

HRESULT Reserved2(
void
);

HRESULT Reserved3(
void
);

HRESULT Reserved4(
void
);

HRESULT Reserved5(
void
);

HRESULT Reserved6(
void
);

HRESULT Reserved7(
void
);

HRESULT EnumTasks( [in, out] unsigned long *taskCount,
    [out, size_is(*taskCount)] TASK_INFO
**taskList);

HRESULT GetTaskDetail([in] LdmObjectId id,
    [in, out] TASK_INFO *tinfo);

HRESULT AbortTask([in] LdmObjectId id);

HRESULT HrGetErrorData( [in] HRESULT hr,
    [in] DWORD dwFlags,
    [out] DWORD *pdwStoredFlags,
    [out] int * pcszw,

```

```

[out, string, size_is(*pcszw,)] wchar_t
*** prgszw );

HRESULT Initialize([in] IUnknown *notificationInterface,
                  [out] unsigned long *ulIDLVersion,
                  [out] DWORD *pdwFlags,
                  [out] LdmObjectId * clientId,
                  [in] unsigned long cRemote);

HRESULT Uninitialize();

HRESULT Refresh();
HRESULT RescanDisks();
HRESULT RefreshFileSys();

HRESULT SecureSystemPartition();
HRESULT ShutDownSystem();

HRESULT EnumAccessPath([in, out] int *lCount,
                      [out, size_is(*lCount)] COUNTED_STRING
**paths);
HRESULT EnumAccessPathForVolume([in] LdmObjectId VolumeId,
                                [in, out] int *lCount,
                                [out, size_is(*lCount)]
COUNTED_STRING **paths);
HRESULT AddAccessPath([in] int cch_path,
                     [in, size_is (cch_path)] WCHAR *path,
                     [in] LdmObjectId targetId);
HRESULT DeleteAccessPath([in] LdmObjectId volumeId,
                        [in] int cch_path,
                        [in, size_is (cch_path)] WCHAR *path);
}

[ object, uuid(DEB01010-3A37-4d26-99DF-E2BB6AE3AC61),
pointer_default(unique) ]
interface IVolumeClient4 : IUnknown
{
    HRESULT RefreshEx( void );

    HRESULT GetVolumeDeviceName(
[in] LdmObjectId          volumeId,
[out] unsigned long*cchVolumeDevice,
                        [out, size_is( ,*cchVolumeDevice)] WCHAR
**pwszVolumeDevice
);
}

```

6.3 unknown.idl

```

typedef GUID IID;

typedef IID *REFIID;

```

```

[
    local,
    object,
    uuid(00000000-0000-0000-C000-000000000046),

    pointer_default(unique)
]

interface IUnknown
{
    typedef [unique] IUnknown *LPUNKNOWN;

    HRESULT QueryInterface(
        [in] REFIID riid,
        [out, iid_is(riid)] void **ppvObject);
    unsigned long AddRef();
    unsigned long Release();
}

```

7 Appendix B: Windows Behavior

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

- Windows Server 2003
- Windows 2000 Server
- Windows XP

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 1.3:](#) The server end of the Disk Management Remote Protocol is implemented by the Windows DmAdmin service on any machine that exposes storage objects for management. On each Windows machine, the client end of the Disk Management Remote Protocol is implemented by the DmRemote COM server. The DmRemote COM server invokes method calls on the DCOM interface on behalf of a number of components, including the Windows Logical Disk Manager (UI) and the Diskpart.exe command-line tool.

[<2> Section 1.4:](#) The Disk Management Remote Protocol is used by the Windows Logical Disk Manager UI and the Diskpart.exe command-line tool.

[<3> Section 1.6:](#) Windows Server 2003, Windows XP, and Windows 2000 implement the Disk Management Remote Protocol. The Windows Server 2003 operating system and Windows Vista implement the [VDS Remote Protocol](#). The [VDS Remote Protocol](#) SHOULD be used with Windows Server 2003. The interfaces associated with the Disk Management Remote Protocol are not available on Windows Vista. The Disk Management Remote Protocol is replaced in Windows Vista by the [VDS Remote Protocol](#).

[<4> Section 1.7:](#)

- The IDMNotify and IDMRemoteServer interfaces are available on Windows Server 2003, Windows 2000 Server, and Windows XP.
- The IVolumeClient and IVolumeClient2 interfaces MUST be used with Windows 2000 Server, and MAY be used with Windows Server 2003 and Windows XP.
- The IVolumeClient3 interface MAY be used with Windows Server 2003 and Windows XP, and MUST NOT be used with Windows 2000 Server.
- The IVolumeClient4 interface MAY be used with Windows Server 2003 and MUST NOT be used with Windows 2000 Server or Windows XP.

[<5> Section 2.1:](#) Windows configures the underlying RPC transport with the following flags, as specified in [\[C706\]](#) and [\[MS-RPCE\]](#):

Interface	Windows 2000 Server	Windows XP (pre-SP2)	Windows Server 2003 and Windows XP SP2
IVolumeClient , IVolumeClient 2, IVolumeClient	RPC_C_AUTHN_LEVEL_CONNECT RPC_C__IMP_LEVEL_IDENTIFY	RPC_C_AUTHN_LEVEL_CONNECT RPC_C__IMP_LEVEL_IMPERSONATE	RPC_C_AUTHN_LEVEL_PKT_PRIVACY RPC_C__IMP_LEVEL_IDENTIFY

Interface	Windows 2000 Server	Windows XP (pre-SP2)	Windows Server 2003 and Windows XP SP2
3, IVolumeClient4	EOAC_APPID	EOAC_NONE	EOAC_SECURE_REFS EOAC_DISABLE_AAA EOAC_NO_CUSTOM_MARSHAL
IDMRemoteServer	RPC_C_AUTHN_LEVEL_NONE RPC_C__IMP_LEVEL_IMPERSONATE EOAC_NONE	RPC_C_AUTHN_LEVEL_NONE RPC_C__IMP_LEVEL_IMPERSONATE EOAC_NONE	RPC_C_AUTHN_LEVEL_NONE RPC_C__IMP_LEVEL_IDENTIFY EOAC_NO_CUSTOM_MARSHALL

<6> Section 2.1: The authorization constraints in Windows vary by operating system release. The table below describes the variations. The boxes of the matrix identify a Windows security group that has the required level of access.

Interface	Windows 2000 Server	Windows XP	Windows Server 2003
IVolumeClient, IVolumeClient2, IVolumeClient3, IVolumeClient4 Launch	Administrators	Administrators, backup operators	Administrators, backup operators Local_system
IVolumeClient, IVolumeClient2, IVolumeClient3, IVolumeClient4 Access	Administrators	Administrators, backup operators	Administrators, backup operators Local_system
IDMRemoteServer Launch	Administrators	Administrators	Administrators Local_system
IDMRemoteServer Access	No restrictions	No restrictions	No restrictions

<7> Section 2.3.1.1: Other OEM partition types recognized by Windows NT 4.0, Windows NT 3.51, Windows NT 3.5, and Windows NT 3.1:

Partition type	Value	Description
PARTITION_EISA	0x12	EISA partition
PARTITION_HIBERNATION	0x84	Hibernation partition for laptops
PARTITION_DIAGNOSTIC	0xA0	Diagnostic partition on some HP notebooks
PARTITION_DELL	0xDE	Dell partition
PARTITION_IBM	0xFE	IBM IML partition

<8> Section 2.5.1.2: Disk signatures are guaranteed to be unique among disks on a single machine.

<9> Section 2.5.1.2: GUIDs generated are guaranteed to be globally unique.

<10> Section 2.5.1.3: Windows 2000 Server and Windows XP servers do not define any partition flags and always initialize this field to 0.

<11> [Section 2.5.1.3:](#) Hidden volumes are not accessible by opening a handle to the file system on the volume using the Win32 API. The volume may be accessed only by opening a handle to the volume device.

<12> [Section 3.1.4:](#) Gaps in the opnum numbering sequence apply to Windows as shown in the following table.

Opnum	Description
IVolumeClient Opnum 0	Default DCOM method QueryInterface
IVolumeClient Opnum 1	Default DCOM method AddRef
IVolumeClient Opnum 2	Default DCOM method Release
IVolumeClient Opnum 12	Used only locally by Windows, never remotely.
IVolumeClient Opnum 27	Not implemented
IVolumeClient Opnum 42	Not implemented
IVolumeClient Opnum 49	Not implemented
IVolumeClient Opnum 50	Not implemented
IVolumeClient Opnum 60	Used only locally by Windows, never remotely.
IVolumeClient Opnum 61	Used only locally by Windows, never remotely.
IVolumeClient Opnum 62	Used only locally by Windows, never remotely.
IVolumeClient Opnum 63	Used only locally by Windows, never remotely.
IVolumeClient Opnum 64	Used only locally by Windows, never remotely.
IVolumeClient Opnum 65	Used only locally by Windows, never remotely.
IVolumeClient Opnum 66	Used only locally by Windows, never remotely.
IVolumeClient2 Opnum 0	Default DCOM method QueryInterface
IVolumeClient2 Opnum 1	Default DCOM method AddRef
IVolumeClient2 Opnum 2	Default DCOM method Release
IVolumeClient3 Opnum 0	Default DCOM method QueryInterface
IVolumeClient3 Opnum 1	Default DCOM method AddRef
IVolumeClient3 Opnum 2	Default DCOM method Release
IVolumeClient3 Opnum 56	Used only locally by Windows, never remotely.
IVolumeClient3 Opnum 57	Used only locally by Windows, never remotely.
IVolumeClient3 Opnum 58	Used only locally by Windows, never remotely.
IVolumeClient3 Opnum 59	Used only locally by Windows, never remotely.
IVolumeClient3 Opnum 60	Used only locally by Windows, never remotely.

Opnum	Description
IVolumeClient3 Opnum 61	Used only locally by Windows, never remotely.
IVolumeClient3 Opnum 62	Used only locally by Windows, never remotely.
IVolumeClient3 Opnum 63	Used only locally by Windows, never remotely.
IVolumeClient4 Opnum 0	Default DCOM method QueryInterface
IVolumeClient4 Opnum 1	Default DCOM method AddRef
IVolumeClient4 Opnum 2	Default DCOM method Release
IDMRemoteServer Opnum 0	Default DCOM method QueryInterface
IDMRemoteServer Opnum 1	Default DCOM method AddRef
IDMRemoteServer Opnum 2	Default DCOM method Release
IDMNotify Opnum 0	Default DCOM method QueryInterface
IDMNotify Opnum 1	Default DCOM method AddRef
IDMNotify Opnum 2	Default DCOM method Release

[<13> Section 3.1.4.1.4:](#) Call sequencing is determined by the invoking application; whether or not this is done is application-specific.

[<14> Section 3.1.4.3:](#) The Disk Management UI client updates the GUI display based on these notifications.

[<15> Section 3.2.4:](#) Windows servers enforce authorization checks. For more information on the authorization requirements for the various methods, see section [2.1](#).

[<16> Section 3.2.4.3:](#) On Windows, all the methods listed that MAY be implemented asynchronously are implemented as asynchronous methods.

[<17> Section 3.2.4.3:](#) For example, on Windows, the call to the file system to format will call back to the server with notifications based on the percentage of the format completed. On Windows, the server sends notifications based on 10-percent increments, so notifications are sent at 0-percent completed, 10-percent completed, 20-percent completed, for example.

[<18> Section 3.2.4.4.1.3:](#) Not used.

[<19> Section 3.2.4.4.1.3:](#) Not used.

[<20> Section 3.2.4.4.1.4:](#) Not used.

[<21> Section 3.2.4.4.1.4:](#) Not used.

[<22> Section 3.2.4.4.1.6:](#) Not used.

[<23> Section 3.2.4.4.1.6:](#) Not used.

[<24> Section 3.2.4.4.1.6:](#) If the force parameter is not set, the call will fail with LDM_E_VOLUME_IN_USE if the volume cannot be locked.

[<25> Section 3.2.4.4.1.7:](#) Not used.

[<26> Section 3.2.4.4.1.7:](#) Not used.

[<27> Section 3.2.4.4.1.8:](#) Not used.

[<28> Section 3.2.4.4.1.8:](#) Not used.

[<29> Section 3.2.4.4.1.9:](#) Not used.

[<30> Section 3.2.4.4.1.9:](#) Not used.

[<31> Section 3.2.4.4.1.12:](#) If the *force* parameter is not set, the call will fail with LDM_E_VOLUME_IN_USE if the volume cannot be locked.

[<32> Section 3.2.4.4.1.12:](#) Not used.

[<33> Section 3.2.4.4.1.12:](#) Not used.

[<34> Section 3.2.4.4.1.13:](#) If the *bForce* parameter is not set, the call will fail with LDM_E_VOLUME_IN_USE if the volume cannot be locked when removing the drive letter associated with the volume.

[<35> Section 3.2.4.4.1.13:](#) Not used.

[<36> Section 3.2.4.4.1.13:](#) Not used.

[<37> Section 3.2.4.4.1.13:](#) Note that the new volume that results when breaking a single volume into two separate partitions may automatically get a new drive letter assigned by the operating system.

[<38> Section 3.2.4.4.1.14:](#) Not used.

[<39> Section 3.2.4.4.1.14:](#) Not used.

[<40> Section 3.2.4.4.1.15:](#) Not used.

[<41> Section 3.2.4.4.1.15:](#) Not used.

[<42> Section 3.2.4.4.1.16:](#) Not used.

[<43> Section 3.2.4.4.1.16:](#) Not used.

[<44> Section 3.2.4.4.1.17:](#) Not used.

[<45> Section 3.2.4.4.1.17:](#) Not used.

[<46> Section 3.2.4.4.1.19:](#) Not used.

[<47> Section 3.2.4.4.1.19:](#) Not used.

[<48> Section 3.2.4.4.1.20:](#) If the *forceOption* parameter is not set, the call will fail with LDM_E_VOLUME_IN_USE if the volume cannot be locked.

[<49> Section 3.2.4.4.1.20:](#) Not used.

[<50> Section 3.2.4.4.1.20:](#) Not used.

[<51> Section 3.2.4.4.1.23:](#) If the *force* parameter is not set, the call will fail with LDM_E_VOLUME_IN_USE if the volume cannot be locked.

[<52> Section 3.2.4.4.1.23:](#) Not used.

[<53> Section 3.2.4.4.1.23:](#) Not used.

[<54> Section 3.2.4.4.1.26:](#) Not used.

[<55> Section 3.2.4.4.1.26:](#) Not used.

[<56> Section 3.2.4.4.1.27:](#) Not used.

[<57> Section 3.2.4.4.1.27:](#) Not used.

[<58> Section 3.2.4.4.1.30:](#) Not used.

[<59> Section 3.2.4.4.1.30:](#) Not used.

[<60> Section 3.2.4.4.1.30:](#) If the *force* parameter is not set, the call will fail with LDM_E_VOLUME_IN_USE if the volume cannot be locked.

[<61> Section 3.2.4.4.1.31:](#) Not used.

[<62> Section 3.2.4.4.1.31:](#) Not used.

[<63> Section 3.2.4.4.1.31:](#) If the *force* parameter is not set, the call will fail with LDM_E_VOLUME_IN_USE if the volume cannot be locked.

[<64> Section 3.2.4.4.1.32:](#) Not used.

[<65> Section 3.2.4.4.1.32:](#) Not used.

[<66> Section 3.2.4.4.1.33:](#) Not used.

[<67> Section 3.2.4.4.1.33:](#) Not used.

[<68> Section 3.2.4.4.1.34:](#) Not used.

[<69> Section 3.2.4.4.1.34:](#) Not used.

[<70> Section 3.2.4.4.1.34:](#) The call fails with LDM_E_VOLUME_IN_USE if the volume cannot be locked.

[<71> Section 3.2.4.4.1.35:](#) Not used.

[<72> Section 3.2.4.4.1.35:](#) Not used.

[<73> Section 3.2.4.4.1.36:](#) Not used.

[<74> Section 3.2.4.4.1.36:](#) Not used.

[<75> Section 3.2.4.4.1.37:](#) Not used.

[<76> Section 3.2.4.4.1.37:](#) Not used.

[<77> Section 3.2.4.4.1.38:](#) Not used.

[<78> Section 3.2.4.4.1.38:](#) Not used.

[<79> Section 3.2.4.4.1.39:](#) Handling of DSKMERGE_IN_NO_UNRELATED flag is not implemented on Windows.

[<80> Section 3.2.4.4.1.40:](#) Not used.

[<81> Section 3.2.4.4.1.40:](#) Not used.

[<82> Section 3.2.4.4.1.41:](#) Not used.

[<83> Section 3.2.4.4.1.41:](#) Not used.

[<84> Section 3.2.4.4.1.42:](#) This method is not implemented. Calls to this method will return E_NOTIMPL.

[<85> Section 3.2.4.4.1.42:](#) Not used.

[<86> Section 3.2.4.4.1.42:](#) Not used.

[<87> Section 3.2.4.4.1.43:](#) Not used.

[<88> Section 3.2.4.4.1.43:](#) Not used.

[<89> Section 3.2.4.4.1.44:](#) Not used.

[<90> Section 3.2.4.4.1.44:](#) Not used.

[<91> Section 3.2.4.4.1.46:](#) Not used.

[<92> Section 3.2.4.4.1.46:](#) Not used.

[<93> Section 3.2.4.4.1.46:](#) This information is used to update the boot.ini file's arcpath for the boot volume. Windows stores these values in the registry under "HKLM\SYSTEM\CurrentControlSet\Services\dmio\Partition Info" as **DWORD** values "OldPartitionNumber" and "NewPartitionNumber."

[<94> Section 3.2.4.4.1.49:](#) The **SetDontShow** method sets a Boolean value that indicates whether to show a disk initialization tool. For more information about this Boolean value, see **GetDontShow**.

[<95> Section 3.2.4.4.1.50:](#) The **GetDontShow** method retrieves a Boolean value that indicates whether to show a disk initialization tool. The New Disk Wizard is part of the UI implementation for Disk Management for Windows. If it is enabled, the wizard appears when the UI is started and uninitialized or empty basic disks are available. Windows servers check a registry value and enable or disable the New Disk Wizard accordingly. The **SetDontShow** method sets the current state of the Boolean value in the registry.

[<96> Section 3.2.4.4.1.54:](#) Windows does not implement this method.

[<97> Section 3.2.4.4.1.54:](#) No flags are defined or returned.

[<98> Section 3.2.4.4.1.54:](#) Windows does not return this information from the server. Windows clients on Windows XP and Windows 2000 make this call but do not depend on it. If the call is not implemented, these clients will print the error information based on the **HRESULT** using strings they retrieve from the binary.

[<99> Section 3.2.4.4.1.55:](#) On Windows, the Logical Disk Manager UI client checks the value of *ulIDLVersion* to be equal to the version of IDL file with which client was built and will disconnect from the server if the *ulIDLVersion* is not the same.

[<100> Section 3.2.4.4.1.55:](#) This flag is never set by Windows Server 2003, Windows 2000 Server, or Windows XP.

[<101> Section 3.2.4.4.1.55:](#) This flag is never set by Windows Server 2003, Windows 2000 Server, or Windows XP.

[<102> Section 3.2.4.4.1.55:](#) This flag is set only by Windows 2000 Server.

[<103> Section 3.2.4.4.1.55:](#) If *cRemote* parameter is nonzero, the server uses server machine account authentication to make calls to the [IDMNotify](#) interface that is specified by *notificationInterface*.

[<104> Section 3.2.4.4.1.60:](#) There is no operating system support for this method, so it is not used by Windows.

[<105> Section 3.2.4.4.3: **IVolumeClient3**](#) methods are not implemented in Windows 2000 Server.

[<106> Section 3.2.4.4.3.7:](#) Not used.

[<107> Section 3.2.4.4.3.7:](#) Not used.

[<108> Section 3.2.4.4.3.32:](#) Not used.

[<109> Section 3.2.4.4.3.32:](#) Not used.

[<110> Section 3.2.4.4.3.33:](#) Not used.

[<111> Section 3.2.4.4.3.33:](#) Not used.

[<112> Section 3.2.4.4.3.38:](#) Not used.

[<113> Section 3.2.4.4.3.38:](#) Not used.

[<114> Section 3.2.4.4.3.48:](#) Not used.

[<115> Section 3.2.4.4.3.48:](#) Not used.

[<116> Section 3.2.4.4.3.51:](#) The [SetDontShow](#) method sets a Boolean value that indicates whether or not to show a disk initialization tool. For more information on this Boolean value, see [GetDontShow \(section 3.2.4.4.3.52\)](#).

[<117> Section 3.2.4.4.3.52:](#) The [GetDontShow](#) method retrieves a value that indicates whether show a disk initialization tool. The New Disk Wizard is part of the UI implementation for Disk Management for Windows. If it is enabled, the wizard appears when the UI is started and uninitialized or empty basic disks are available. Windows servers check a registry value and enable or disable the New Disk Wizard accordingly. The [SetDontShow](#) method sets the current state of the Boolean value in the registry.

[<118> Section 3.2.4.4.3.56:](#) No flags are defined, and this parameter is always initialized to 0.

[<119> Section 3.2.4.4.3.57:](#) This flag is never set by Windows Server 2003, Windows 2000 Server, or Windows XP.

[<120> Section 3.2.4.4.3.57:](#) This flag is never set by Windows Server 2003, Windows 2000 Server, or Windows XP.

[<121> Section 3.2.4.4.3.57:](#) This flag is set only by Windows 2000 Server.

[<122> Section 3.2.4.4.4: **IVolumeClient4**](#) methods are not implemented in Windows 2000 Server or Windows XP.

[<123> Section 3.2.4.4.1:](#) The Windows volume manager keeps track of the dynamic disks present on a system and displays disks that are no longer present as missing.

[<124> Section 3.2.6:](#) Windows servers do register for such notifications.

8 Appendix C: IDMNotify::ObjectsChanged

```
void CClientClass::ObjectsChanged( DWORD dwByteCount, BYTE *pByte)
{
    DWORD    dwNotifSize    = 0L;
    DWORD    dwCopySize    = 0L;
    BYTE     *pPos          = pByte;
    BYTE     *pNotifStart   = NULL;

    DMNOTIFY_INFO_TYPE    Type;
    LDMACTION              Action;

    while (pPos - pByte < (long)dwByteCount)
    {
        pNotifStart = pPos;

        // Get the notification size.
        memcpy( &dwNotifSize, pPos, sizeof(DWORD) );
        pPos = pPos + sizeof(DWORD);

        // Get the notification type.
        memcpy( &Type, pPos, sizeof(DMNOTIFY_INFO_TYPE) );
        pPos = pPos + sizeof(DMNOTIFY_INFO_TYPE);

        // Get the notification action.
        memcpy( &Action, pPos, sizeof(LDMACTION) );
        pPos = pPos + sizeof(LDMACTION);

        // dwCopySize is the number of bytes left to copy out
        // of the byte stream for this notification.
        dwCopySize = dwNotifSize
            - ( sizeof(DWORD)
              + sizeof(DMNOTIFY_INFO_TYPE)
              + sizeof(LDMACTION) );

        // Switch on the type of this notification.
        switch (Type) {

        case DMNOTIFY_DISK_INFO:

            // We need to treat IVolumeClient server and IVolumeClient3
            // server differently. IVolumeClient server uses DISK_INFO,
            // IVolumeClient3 uses DISK_INFO_EX. The code below will load
            // DISK_INFO into a DISK_INFO_EX structure for the case where
            // the server is Windows 2000 and the client is Windows XP or
            // Windows 2003.

            DISK_INFO_EX DiskInfoEx;

            memset(&DiskInfoEx, 0, sizeof(DISK_INFO_EX));

            if ( nIVolumeClientVersion == 3 )
            {
                dwCopySize = offsetof(DISK_INFO_EX, name);

                memcpy(&DiskInfoEx, pPos, offsetof(DISK_INFO_EX, name));
                pPos = pPos + dwCopySize;
            }
        }
    }
}
```

```

}
else    // nIVolumeClientVersion == 1
{
    //
    // Copy the first part of disk info structure.
    //

    DISK_INFO DiskInfo;

    memset(&DiskInfo,0,sizeof(DISK_INFO));

    //
    // On a 64-bit client, 4 bytes of padding are added
    // after cchDgName, so we cannot set dwCopySize to
    // offsetof(DISK_INFO,name) - the byte stream passed
    // from the 32-bit server does not have these 4 bytes
    // of padding.
    //
    // However, if the client is 32 bit and the server is
    // 64 bit, the code below sets pPos to point to the
    // padding.
    //
    // dwCopySize = offsetof(DISK_INFO,cchDgName)
    //             + sizeof(DiskInfo.cchDgName);
    // memcpy( &DiskInfo, pPos, dwCopySize );
    // pPos = pPos + dwCopySize;    // pPos may now be
    //                               // incorrect.
    //
    // We have this problem below in the DMNOTIFY_FS_INFO
    // case.
    //
    // The workaround is to setup globals that store the
    // server and client architecture. For the client, call
    // the Win32 API GetSystemInfo().
    //
    // For the server, use the Disk Management interfaces
    // to look for an ESP partition on any of the client
    // disks. If one is found, assume a 64-bit
    // architecture. For this code, assume
    // g_ClientArchitecture and g_ServerArchitecture have
    // been setup.
    //

    if ( g_ClientArchitecture == g_ServerArchitecture ) {
        dwCopySize = offsetof(DISK_INFO, name)
        memcpy( &DiskInfo, pPos, dwCopySize );
        pPos = pPos + dwCopySize;
    }
    else if ( g_ClientArchitecture == 32 bit ) {
        dwCopySize = offsetof(DISK_INFO,name);
        memcpy( &DiskInfo, pPos, dwCopySize );
        pPos = pPos + dwCopySize;
    }
    else {    // ( g_ServerArchitecture == 32 bit )
        dwCopySize = offsetof(DISK_INFO,cchDgName)
        + sizeof(DiskInfo.cchDgName);
        memcpy( &DiskInfo, pPos, dwCopySize );
    }
}

```



```

        pPos = pPos + dwCopySize;
    }

    //
    // Copy from DISK_INFO to DISK_INFO_EX
    //

    CopyToDiskInfoEx( &DiskInfo, &DiskInfoEx );
}

// Copy disk name.
wchar_t *name;
name = new wchar_t[DiskInfoEx.cchName * sizeof(wchar_t)];
if (name)
    memcpy( name,
            pPos,
            sizeof(wchar_t) * DiskInfoEx.cchName );
pPos = pPos + (sizeof(wchar_t) * DiskInfoEx.cchName);

// Copy disk vendor.
wchar_t *vendor;
vendor = new wchar_t[DiskInfoEx.cchVendor
    * sizeof(wchar_t)];
if (vendor)
    memcpy( vendor,
            pPos,
            sizeof(wchar_t) * DiskInfoEx.cchVendor );
pPos = pPos + (sizeof(wchar_t) * DiskInfoEx.cchVendor);

// Copy disk group id.
BYTE *dgid;
dgid = new BYTE[DiskInfoEx.cchDgid * sizeof(BYTE)];
if (dgid)
    memcpy( dgid,
            pPos,
            sizeof(BYTE) * DiskInfoEx.cchDgid );
pPos = pPos + (sizeof(BYTE) * DiskInfoEx.cchDgid);

// Copy disk adapter.
wchar_t *adapterName;
adapterName = new wchar_t[DiskInfoEx.cchAdapterName
    * sizeof(wchar_t)];
if (adapterName)
    memcpy( adapterName,
            pPos,
            sizeof(wchar_t) * DiskInfoEx.cchAdapterName );
pPos = pPos
    + (sizeof(wchar_t) * DiskInfoEx.cchAdapterName);

// Copy disk group name.
wchar_t *dgName;
dgName = new wchar_t[DiskInfoEx.cchDgName
    * sizeof(wchar_t)];
if (dgName)
    memcpy( dgName,
            pPos,
            sizeof(wchar_t) * DiskInfoEx.cchDgName );
pPos = pPos + (sizeof(wchar_t) * DiskInfoEx.cchDgName);

```

```

        // Copy device instance id.
        wchar_t *devInstId;

        if ( nIVolumeClientVersion == 3 )
        {
            // Copy device instance id.
            if (DiskInfoEx.cchDevInstId)
            {
                devInstId = new wchar_t[DiskInfoEx.cchDevInstId *
                    sizeof(wchar_t)];
                if (devInstId)
                    memcpy( devInstId, pPos, sizeof(wchar_t) *
                        DiskInfoEx.cchDevInstId );
                pPos = pPos + (sizeof(wchar_t) *
                    DiskInfoEx.cchDevInstId);
            }
            else
                devInstId = NULL;
        }
        else // nIVolumeClientVersion == 1
            devInstId = NULL;

        //
        // Assign the rest of the DISK_INFO_EX members.
        //
        DiskInfoEx.name = name;
        DiskInfoEx.vendor = vendor;
        DiskInfoEx.dgid = dgid;
        DiskInfoEx.adapterName = adapterName;
        DiskInfoEx.dgName = dgName;
        DiskInfoEx.devInstId = devInstId;

        break;

    case DMNOTIFY_VOLUME_INFO:

        VOLUME_INFO VolumeInfo;

        memset( &VolumeInfo, 0, sizeof(VOLUME_INFO) );

        // Copy in volume info.

        memcpy( &VolumeInfo, pPos, dwCopySize );
        pPos = pPos + dwCopySize;

        break;

    case DMNOTIFY_REGION_INFO:

        REGION_INFO_EX RegInfoEx;

        memset( &RegInfoEx, 0, sizeof(REGION_INFO_EX) );

        // We need to treat IVolumeClient server and IVolumeClient3
        // server differently. IVolumeClient server uses
        // REGION_INFO instead of REGION_INFO_EX. The code below
        // will load REGION_INFO into a REGION_INFO_EX structure

```

```

// for the case where the server is Windows 2000 and the
// client is Windows XP or Windows 2003

if ( nIVolumeClientVersion==3 )
{
    memcpy( &RegInfoEx,
            pPos,
            offsetof(REGION_INFO_EX,name) );
    pPos += offsetof(REGION_INFO_EX,name);

    // Copy name.
    wchar_t *name;
    name = new wchar_t[RegInfoEx.cchName
        * sizeof(wchar_t)];
    if (name)
        memcpy( name,
                pPos,
                sizeof(wchar_t) * RegInfoEx.cchName );
    pPos = pPos + (sizeof(wchar_t) * RegInfoEx.cchName);

    RegInfoEx.name = name;
}
else // m_sIVolumeClientVersion == 1
{
    REGION_INFO RegInfo;

    memset( &RegInfo, 0, sizeof(REGION_INFO) );

    memcpy( &RegInfo, pPos, dwCopySize );
    pPos = pPos + dwCopySize;

    CopyToRegionInfoEx( &RegInfo, &RegInfoEx );
}
break;

case DMNOTIFY_TASK_INFO:

    TASK_INFO TaskInfo;

    memset( &TaskInfo, 0, sizeof(TASK_INFO) );

    // Copy in task info.

    memcpy( &TaskInfo, pPos, dwCopySize );
    pPos = pPos + dwCopySize;

    break;

case DMNOTIFY_DL_INFO:

    DRIVE_LETTER_INFO DLInfo;

    memset( &DLInfo, 0, sizeof(DRIVE_LETTER_INFO) );

    // Copy in drive letter info.

    memcpy( &DLInfo, pPos, dwCopySize );
    pPos = pPos + dwCopySize;

```

```

        break;

case DMNOTIFY_FS_INFO:

    FILE_SYSTEM_INFO FsInfo;

    memset( &FsInfo, 0, sizeof(FILE_SYSTEM_INFO) );

    //
    // We have this problem here as above in the
    // DMNOTIFY_DISK_INFO case.
    //
    // On a 64-bit client, 4 bytes of padding are added after
    // cchLabel, so we cannot set dwCopySize to
    // offsetof(FILE_SYSTEM_INFO, label) - the byte stream passed
    // from the 32-bit server does not have these 4 bytes of
    // padding.
    //
    // However, if the client is 32 bit and the server is 64
    // bit, the code below sets pPos to point to the padding.
    //
    // dwCopySize = offsetof(FILE_SYSTEM_INFO, cchLabel)
    //               + sizeof(FsInfo.cchLabel);
    // memcpy( &FsInfo, pPos, dwCopySize );
    // pPos = pPos + dwCopySize; // pPos may now be
    //                          // incorrect.
    //

    // Copy file system info.

    if ( g_ClientArchitecture == g_ServerArchitecture ) {
        dwCopySize = offsetof(FILE_SYSTEM_INFO, label)
        memcpy( &DiskInfo, pPos, dwCopySize );
        pPos = pPos + dwCopySize;
    }
    else if ( g_ClientArchitecture == 32 bit ) {
        dwCopySize = offsetof(FILE_SYSTEM_INFO, label);
        memcpy( &DiskInfo, pPos, dwCopySize );
        pPos = pPos + dwCopySize;
    }
    else { // ( g_ServerArchitecture == 32 bit )
        dwCopySize = offsetof(FILE_SYSTEM_INFO, cchLabel)
        + sizeof(FsInfo.cchLabel);
        memcpy( &DiskInfo, pPos, dwCopySize );
        pPos = pPos + dwCopySize;
    }

    // Copy the label.
    wchar_t *label;
    label = new wchar_t[FsInfo.cchLabel * sizeof(wchar_t)];
    if (label)
        memcpy( label,
            pPos,
            sizeof(wchar_t) * FsInfo.cchLabel );
    pPos = pPos + (sizeof(wchar_t) * FsInfo.cchLabel);
    FsInfo.label = label;

```

```

        break;

    case DMNOTIFY_SYSTEM_INFO:

        DWORD SysInfo;

        memset( &SysInfo, 0, sizeof(DWORD) );

        // Copy in system info.

        memcpy( &SysInfo, pPos, dwCopySize );
        pPos = pPos + dwCopySize;

        break;

    } // switch

    pPos = pNotifStart + dwNotifSize;
} // while

}

```

9 Index

[_PARTITIONSTYLE enumeration](#)

A

AbortTask method ([section 3.2.4.4.1.53](#), [section 3.2.4.4.3.55](#))
Abstract data model
 [client](#)
 [server](#)
AddAccessPath method ([section 3.2.4.4.1.64](#), [section 3.2.4.4.3.66](#))
AddMirror method ([section 3.2.4.4.1.32](#), [section 3.2.4.4.3.35](#))
[Applicability](#)
AssignDriveLetter method ([section 3.2.4.4.1.19](#), [section 3.2.4.4.3.19](#))

C

[Capability negotiation](#)
Client
 [abstract data model](#)
 [higher-layer triggered events](#)
 [local events](#)
 [message processing](#)
 [overview](#)
 [sequencing rules](#)
 [timer events](#)
 [timers](#)
[Common data types](#)
[Connection loss](#)
Connections ([section 3.2.1.2](#), [section 3.2.3.2](#))
[COUNTED_STRING structure](#)
CreatePartition method ([section 3.2.4.4.1.3](#), [section 3.2.4.4.3.3](#))
CreatePartitionAssignAndFormat method ([section 3.2.4.4.1.4](#), [section 3.2.4.4.3.4](#))
CreatePartitionAssignAndFormatEx method ([section 3.2.4.4.1.5](#), [section 3.2.4.4.3.5](#))
[CreatePartitionsForVolume method](#)
[CreateRemoteObject method](#)
CreateVolume method ([section 3.2.4.4.1.26](#), [section 3.2.4.4.3.26](#))
CreateVolumeAssignAndFormat method ([section 3.2.4.4.1.27](#), [section 3.2.4.4.3.27](#))
CreateVolumeAssignAndFormatEx method ([section 3.2.4.4.1.28](#), [section 3.2.4.4.3.28](#))

D

Data model - abstract
 [client](#)
 [server](#)
Data types
 [common](#)
 [IDMNotify interface](#)
 [IDMRemoteServer interface](#)
 [IVolumeClient interface](#)
 [IVolumeClient2 interface](#)

[IVolumeClient3 interface](#)

[IVolumeClient4 interface](#)

DeleteAccessPath method ([section 3.2.4.4.1.65](#), [section 3.2.4.4.3.67](#))
DeletePartition method ([section 3.2.4.4.1.6](#), [section 3.2.4.4.3.6](#))
DeletePartitionNumberInfoFromRegistry method ([section 3.2.4.4.1.48](#), [section 3.2.4.4.3.50](#))
[DeletePartitionsForVolume method](#)
DeleteVolume method ([section 3.2.4.4.1.31](#), [section 3.2.4.4.3.31](#))
[Disk arrival](#)
[Disk layout change](#)
[Disk removal](#)
[DISK_INFO structure](#)
[DISK_INFO_EX structure](#)
[DISK_SPEC structure](#)
DiskMerge method ([section 3.2.4.4.1.40](#), [section 3.2.4.4.3.43](#))
DiskMergeQuery method ([section 3.2.4.4.1.39](#), [section 3.2.4.4.3.42](#))
[Disks](#)
[dmintf.idl](#)
[dmintf3.idl](#)
[DMNotify::ObjectsChanged](#)
[DMNOTIFY_INFO_TYPE enumeration](#)
[dmProgressType enumeration](#)
[Drive letter arrival](#)
[Drive letter removal](#)
[Drive letters](#)
[DRIVE_LETTER_INFO structure](#)

E

Eject method ([section 3.2.4.4.1.9](#), [section 3.2.4.4.3.9](#))
[EncapsulateDisk method](#)
[EncapsulateDiskEx method](#)
EnumAccessPath method ([section 3.2.4.4.1.62](#), [section 3.2.4.4.3.64](#))
EnumAccessPathForVolume method ([section 3.2.4.4.1.63](#), [section 3.2.4.4.3.65](#))
[EnumDiskRegions method](#)
[EnumDiskRegionsEx method](#)
[EnumDisks method](#)
[EnumDisksEx method](#)
EnumDriveLetters method ([section 3.2.4.4.1.18](#), [section 3.2.4.4.3.18](#))
EnumLocalFileSystems method ([section 3.2.4.4.1.21](#), [section 3.2.4.4.3.21](#))
EnumTasks method ([section 3.2.4.4.1.51](#), [section 3.2.4.4.3.53](#))
EnumVolumeMembers method ([section 3.2.4.4.1.25](#), [section 3.2.4.4.3.25](#))
EnumVolumes method ([section 3.2.4.4.1.24](#), [section 3.2.4.4.3.24](#))
[Examples](#)

F

[Fields - vendor-extensible](#)

[File system change](#)

[File systems](#)

[FILE_SYSTEM_INFO structure](#)

[Format method \(section 3.2.4.4.1.23, section 3.2.4.4.3.23\)](#)

[FreeDriveLetter method \(section 3.2.4.4.1.20, section 3.2.4.4.3.20\)](#)

[FTBreakMirror method \(section 3.2.4.4.1.13, section 3.2.4.4.3.13\)](#)

[FTDeleteVolume method \(section 3.2.4.4.1.12, section 3.2.4.4.3.12\)](#)

[FTEnumLogicalDiskMembers method \(section 3.2.4.4.1.11, section 3.2.4.4.3.11\)](#)

[FTEnumVolumes method \(section 3.2.4.4.1.10, section 3.2.4.4.3.10\)](#)

[FTRegenerateParityStripe method \(section 3.2.4.4.1.15, section 3.2.4.4.3.15\)](#)

[FTReplaceMirrorPartition method \(section 3.2.4.4.1.16, section 3.2.4.4.3.16\)](#)

[FTReplaceParityStripePartition method \(section 3.2.4.4.1.17, section 3.2.4.4.3.17\)](#)

[FTResyncMirror method \(section 3.2.4.4.1.14, section 3.2.4.4.3.14\)](#)

[Full IDL \(section 6, section 6.1, section 6.2, section 6.3\)](#)

G

[GetDontShow method \(section 3.2.4.4.1.50, section 3.2.4.4.3.52\)](#)

[GetEncapsulateDiskInfo method](#)

[GetEncapsulateDiskInfoEx method](#)

[GetInstalledFileSystems method \(section 3.2.4.4.1.22, section 3.2.4.4.3.22\)](#)

[GetMaxAdjustedFreeSpace method \(section 3.2.4.4.2.1, section 3.2.4.4.3.34\)](#)

[GetTaskDetail method \(section 3.2.4.4.1.52, section 3.2.4.4.3.54\)](#)

[GetVolumeDeviceName method](#)

[GetVolumeMountName method \(section 3.2.4.4.1.29, section 3.2.4.4.3.29\)](#)

[Glossary](#)

[GrowVolume method \(section 3.2.4.4.1.30, section 3.2.4.4.3.30\)](#)

H

Higher-layer triggered events

[client](#)

[server](#)

[HrGetErrorData method \(section 3.2.4.4.1.54, section 3.2.4.4.3.56\)](#)

I

[IDL \(section 6, section 6.1, section 6.2, section 6.3\)](#)

[IDMNotify interface](#)

[data types](#)

[overview](#)

[IDMNotify methods](#)

[IDMRemoteServer interface](#)

[data types](#)

[overview](#)

[IFILE_SYSTEM_INFO structure](#)

[ImportDiskGroup method \(section 3.2.4.4.1.38, section 3.2.4.4.3.41\)](#)

[Informative references](#)

[Initialization](#)

[client](#)

[server](#)

[Initialize method \(section 3.2.4.4.1.55, section 3.2.4.4.3.57\)](#)

[InitializeDisk method](#)

[InitializeDiskEx method](#)

[InitializeDiskStyle method](#)

[Introduction](#)

[IVolumeClient interface](#)

[data types](#)

[overview \(section 2.3, section 2.5, section 2.6\)](#)

[IVolumeClient2 interface](#)

[data types](#)

[overview](#)

[IVolumeClient3 interface - data types](#)

[IVolumeClient4 interface - data types](#)

L

[LDMACTION enumeration](#)

[List of client connections \(section 3.2.1.2, section 3.2.3.2\)](#)

[List of current tasks \(section 3.2.1.3, section 3.2.3.3\)](#)

[List of storage objects \(section 3.2.1.1, section 3.2.3.1\)](#)

[Local events](#)

[client](#)

[server](#)

[Loss of connection](#)

M

[MarkActivePartition method \(section 3.2.4.4.1.8, section 3.2.4.4.3.8\)](#)

[MAX_FS_NAME_SIZE](#)

[Media arrival](#)

[Media removal](#)

[MERGE_OBJECT_INFO structure](#)

[Message processing](#)

[client](#)

[server](#)

[Messages](#)

[data types](#)

[details \(section 3.1.4.4, section 3.2.4.4\)](#)

[overview](#)

[transport](#)

[Methods with prerequisites](#)

N

[Normative references](#)

O

[ObjectsChanged method](#)
[Overview \(synopsis\)](#)

P

[Parameters to IVolumeClient and IVolumeClient3](#)
[PARTITION_OS2_BOOT](#)
[Partitions](#)
[Preconditions](#)
[Prerequisites](#)
[Processing notifications - server to client](#)
[Processing server replies to method calls](#)

Q

[QueryChangePartitionNumbers method \(section 3.2.4.4.1.47, section 3.2.4.4.3.49\)](#)

R

[ReAttachDisk method \(section 3.2.4.4.1.41, section 3.2.4.4.3.44\)](#)
[ReConnectDisk method \(section 3.2.4.4.1.37, section 3.2.4.4.3.40\)](#)
[References](#)
[informative](#)
[normative](#)
[overview](#)
[Refresh method \(section 3.2.4.4.1.57, section 3.2.4.4.3.59\)](#)
[RefreshEx method](#)
[RefreshFileSys method \(section 3.2.4.4.1.59, section 3.2.4.4.3.61\)](#)
[REGION_INFO structure](#)
[REGION_INFO_EX structure](#)
[REGION_SPEC structure](#)
[REGIONSTATUS enumeration](#)
[REGIONTYPE enumeration](#)
[Relationship to other protocols](#)
[Relationships between storage objects](#)
[RemoveMirror method \(section 3.2.4.4.1.33, section 3.2.4.4.3.36\)](#)
[ReplaceDisk method](#)
[ReplaceRaid5Column method \(section 3.2.4.4.1.43, section 3.2.4.4.3.45\)](#)
[REQSTATUS enumeration](#)
[RescanDisks method \(section 3.2.4.4.1.58, section 3.2.4.4.3.60\)](#)
[RestartVolume method \(section 3.2.4.4.1.44, section 3.2.4.4.3.46\)](#)
[Rules - asynchronous tasks](#)
[Rules - modify storage objects list](#)
[Rules - synchronous tasks](#)

S

[SecureSystemPartition method \(section 3.2.4.4.1.60, section 3.2.4.4.3.62\)](#)
[Security](#)

[Sequencing rules](#)

[client](#)
[server](#)

[Server](#)

[abstract data model](#)
[higher-layer triggered events](#)
[initialization](#)
[local events](#)
[message processing](#)
[overview](#)
[sequencing rules](#)
[timer events](#)
[timers](#)

[SetDontShow method \(section 3.2.4.4.1.49, section 3.2.4.4.3.51\)](#)

[ShutDownSystem method \(section 3.2.4.4.1.61, section 3.2.4.4.3.63\)](#)

[SplitMirror method \(section 3.2.4.4.1.34, section 3.2.4.4.3.37\)](#)

[Standards assignments](#)

[Storage objects \(section 3.2.1.1, section 3.2.3.1\)](#)

T

[TASK_INFO structure](#)

[Tasks](#)

[Tasks currently executed \(section 3.2.1.3, section 3.2.3.3\)](#)

[Timer events](#)

[client](#)
[server](#)

[Timers](#)

[client](#)
[server](#)

[Transport](#)

[Triggered events - higher-layer](#)

[client](#)
[server](#)

U

[Uninitialize method \(section 3.2.4.4.1.56, section 3.2.4.4.3.58\)](#)

[UninitializeDisk method \(section 3.2.4.4.1.36, section 3.2.4.4.3.39\)](#)
[unknwn.idl](#)

V

[Vendor-extensible fields](#)

[Versioning](#)

[VOLUME_INFO structure](#)

[VOLUME_SPEC structure](#)

[VOLUME_LAYOUT enumeration](#)

[Volumes](#)

[VOLUMESTATUS enumeration](#)

[VOLUMETYPE enumeration](#)

W

[Windows behavior](#)

[WriteSignature method](#)