[MS-DMRP-Diff]:

Disk Management Remote Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards ~~as well as overviews of the interaction among each of these technologies~~support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This 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~~can make copies of it in order to develop implementations of the technologies that are described in ~~the Open Specifications~~ this documentation and ~~may~~can distribute portions of it in your implementations ~~using~~that use these technologies or in your documentation as necessary to properly document the implementation. You ~~may~~can also distribute in your implementation, with or without modification, any ~~schema, IDL's~~schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications~~.~~ documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that ~~may~~might cover your implementations of the technologies described in the Open Specifications~~.~~ documentation. Neither this notice nor Microsoft's delivery of ~~the~~this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open ~~Specification may~~Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in ~~the Open Specifications~~this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks**. The names of companies and products contained in this documentation ~~may~~might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, ~~e-mail~~email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications ~~do~~documentation 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. Certain Open Specifications documents are intended for use in conjunction with publicly available ~~standard~~standards specifications and network programming art~~,~~ and ~~assumes~~, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
3/2/2007	1.0	Major	Updated and revised the technical content.
4/3/2007	1.1	Minor	Clarified the meaning of the technical content.
5/11/2007	2.0	Major	New format; Updated technical content
6/1/2007	2.0.1	Editorial	Changed language and formatting in the technical content.
7/3/2007	3.0	Major	Updated and revised the technical content.
8/10/2007	4.0	Major	Updated and revised the technical content.
9/28/2007	4.0.1	Editorial	Changed language and formatting in the technical content.
10/23/2007	4.1	Minor	Updated the IDL.
1/25/2008	4.1.1	Editorial	Changed language and formatting in the technical content.
3/14/2008	5.0	Major	Updated and revised the technical content.
6/20/2008	6.0	Major	Updated and revised the technical content.
7/25/2008	7.0	Major	Updated and revised the technical content.
8/29/2008	7.1	Minor	Clarified the meaning of the technical content.
10/24/2008	7.2	Minor	Clarified the meaning of the technical content.
12/5/2008	7.3	Minor	Clarified the meaning of the technical content.
1/16/2009	7.4	Minor	Clarified the meaning of the technical content.
2/27/2009	7.5	Minor	Clarified the meaning of the technical content.
4/10/2009	7.5.1	Editorial	Changed language and formatting in the technical content.
5/22/2009	7.5.2	Editorial	Changed language and formatting in the technical content.
7/2/2009	7.5.3	Editorial	Changed language and formatting in the technical content.
8/14/2009	7.5.4	Editorial	Changed language and formatting in the technical content.
9/25/2009	7.6	Minor	Clarified the meaning of the technical content.
11/6/2009	7.6.1	Editorial	Changed language and formatting in the technical content.
12/18/2009	7.6.2	Editorial	Changed language and formatting in the technical content.
1/29/2010	7.7	Minor	Clarified the meaning of the technical content.
3/12/2010	7.7.1	Editorial	Changed language and formatting in the technical content.
4/23/2010	7.7.2	Editorial	Changed language and formatting in the technical content.
6/4/2010	7.7.3	Editorial	Changed language and formatting in the technical content.
7/16/2010	7.7.3	None	No changes to the meaning, language, or formatting of the technical content.

Copyright © 2016 Microsoft Corporation

Date	Revision History	Revision Class	Comments
8/27/2010	8.0	Major	Updated and revised the technical content.
10/8/2010	8.0	None	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	8.0	None	No changes to the meaning, language, or formatting of the technical content.
1/7/2011	8.0	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2011	8.0	None	No changes to the meaning, language, or formatting of the technical content.
3/25/2011	8.0	None	No changes to the meaning, language, or formatting of the technical content.
5/6/2011	8.0	None	No changes to the meaning, language, or formatting of the technical content.
6/17/2011	8.1	Minor	Clarified the meaning of the technical content.
9/23/2011	8.1	None	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	8.1	None	No changes to the meaning, language, or formatting of the technical content.
3/30/2012	8.1	None	No changes to the meaning, language, or formatting of the technical content.
7/12/2012	8.2	Minor	Clarified the meaning of the technical content.
10/25/2012	8.2	None	No changes to the meaning, language, or formatting of the technical content.
1/31/2013	8.2	None	No changes to the meaning, language, or formatting of the technical content.
8/8/2013	8.2	None	No changes to the meaning, language, or formatting of the technical content.
11/14/2013	8.2	None	No changes to the meaning, language, or formatting of the technical content.
2/13/2014	8.2	None	No changes to the meaning, language, or formatting of the technical content.
5/15/2014	8.2	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	8.2	~~No Change~~None	No changes to the meaning, language, or formatting of the technical content.
10/16/2015	8.2	~~No Change~~None	No changes to the meaning, language, or formatting of the technical content.

Copyright © 2016 Microsoft Corporation

Table of Contents

1 Introduction 9
1.1 Glossary 9
1.2 References 15
1.2.1 Normative References 15
1.2.2 Informative References 15
1.3 Overview 16
1.4 Relationship to Other Protocols 17
1.5 Prerequisites/Preconditions 17
1.6 Applicability Statement 17
1.7 Versioning and Capability Negotiation 17
1.8 Vendor-Extensible Fields 17
1.9 Standards Assignments 17

2 Messages 18
2.1 Transport 18
2.2 Common Data Types 18
2.2.1 HRESULT Return Codes 18
2.2.2 MAX_FS_NAME_SIZE Constant 31
2.2.3 REGIONTYPE 32
2.2.4 VOLUMETYPE 32
2.2.5 VOLUMELAYOUT 33
2.2.6 REQSTATUS 33
2.2.7 REGIONSTATUS 34
2.2.8 VOLUMESTATUS 34
2.2.9 LdmObjectId 35
2.2.10 VOLUME_SPEC 35
2.2.11 VOLUME_INFO 35
2.2.12 DISK_SPEC 37
2.2.13 REGION_SPEC 37
2.2.14 DRIVE_LETTER_INFO 38
2.2.15 FILE_SYSTEM_INFO 39
2.2.16 IFILE_SYSTEM_INFO 40
2.2.17 TASK_INFO 42
2.2.18 DMPROGRESS_TYPE 43
2.2.19 COUNTED_STRING 43
2.2.20 MERGE_OBJECT_INFO 44
2.3 IVolumeClient Interface 45
2.3.1 IVolumeClient Data Types 45
2.3.1.1 PARTITION_OS2_BOOT Constant 45
2.3.1.2 DISK_INFO 45
2.3.1.3 REGION_INFO 48
2.4 IVolumeClient2 Interface 50
2.4.1 IVolumeClient2 Data Types 50
2.5 IVolumeClient3 Interface 50
2.5.1 IVolumeClient3 Data Types 50
2.5.1.1 PARTITIONSTYLE 50
2.5.1.2 DISK_INFO_EX 51
2.5.1.3 REGION_INFO_EX 55
2.6 IVolumeClient4 Interface 58
2.6.1 IVolumeClient4 Data Types 58
2.7 IDMRemoteServer Interface 58
2.7.1 IDMRemoteServer Data Types 58
2.8 IDMNotify Interface 58
2.8.1 IDMNotify Data Types 58
2.8.1.1 DMNOTIFY_INFO_TYPE 58

Copyright © 2016 Microsoft Corporation

2.8.1.2 LDMACTION 59

3 Protocol Details 60

3.1 Client Role Details 60
3.1.1 Abstract Data Model 60
3.1.2 Timers 60
3.1.3 Initialization 60
3.1.4 Message Processing and Sequencing Rules 60
3.1.4.1 Higher-Layer Triggered Events 65
3.1.4.1.1 Common Details 65
3.1.4.1.1.1 Methods with Prerequisites 65
3.1.4.1.1.2 Parameters to IVolumeClient and IVolumeClient3 65
3.1.4.1.1.3 Relationships Between Storage Objects 66
3.1.4.1.2 Drive Letters 66
3.1.4.1.3 File Systems 67
3.1.4.1.4 Disks 67
3.1.4.1.5 Partitions 70
3.1.4.1.6 Volumes 70
3.1.4.1.7 Tasks 73
3.1.4.1.8 Loss of Connection 73
3.1.4.2 Processing Server Replies to Method Calls 73
3.1.4.3 Processing Notifications Sent from the Server to the Client 73
3.1.4.4 Protocol Message Details 74
3.1.4.4.1 IDMNotify Methods 74
3.1.4.4.1.1 IDMNotify::ObjectsChanged (Opnum 3) 74
3.1.5 Timer Events 76
3.1.6 Other Local Events 76
3.2 Server Role Details 76
3.2.1 Abstract Data Model 76
3.2.1.1 List of Storage Objects Present in the System 76
3.2.1.2 List of Clients Connected to the Server 77
3.2.1.3 List of Tasks Currently Executed on the Server 78
3.2.2 Timers 78
3.2.3 Initialization 78
3.2.3.1 List of Storage Objects Present in the System 78
3.2.3.2 List of Clients Connected to the Server 78
3.2.3.3 List of Tasks Currently Executed on the Server 78
3.2.4 Message Processing and Sequencing Rules 78
3.2.4.1 Higher-Layer Triggered Events 79
3.2.4.2 Rules for Modifying the List of Storage Objects 79
3.2.4.3 Rules for Handling Synchronous and Asynchronous Tasks 79
3.2.4.4 Protocol Message Details 81
3.2.4.4.1 IVolumeClient Methods 81
3.2.4.4.1.1 IVolumeClient::EnumDisks (Opnum 3) 84
3.2.4.4.1.2 IVolumeClient::EnumDiskRegions (Opnum 4) 85
3.2.4.4.1.3 IVolumeClient::CreatePartition (Opnum 5) 86
3.2.4.4.1.4 IVolumeClient::CreatePartitionAssignAndFormat (Opnum 6) 87
3.2.4.4.1.5 IVolumeClient::CreatePartitionAssignAndFormatEx (Opnum 7) 89
3.2.4.4.1.6 IVolumeClient::DeletePartition (Opnum 8) 90
3.2.4.4.1.7 IVolumeClient::WriteSignature (Opnum 9) 92
3.2.4.4.1.8 IVolumeClient::MarkActivePartition (Opnum 10) 93
3.2.4.4.1.9 IVolumeClient::Eject (Opnum 11) 94
3.2.4.4.1.10 IVolumeClient::FTEnumVolumes (Opnum 13) 95
3.2.4.4.1.11 IVolumeClient::FTEnumLogicalDiskMembers (Opnum 14) 96
3.2.4.4.1.12 IVolumeClient::FTDeleteVolume (Opnum 15) 97
3.2.4.4.1.13 IVolumeClient::FTBreakMirror (Opnum 16) 98
3.2.4.4.1.14 IVolumeClient::FTResyncMirror (Opnum 17) 100
3.2.4.4.1.15 IVolumeClient::FTRegenerateParityStripe (Opnum 18) 101

Copyright © 2016 Microsoft Corporation

3.2.4.4.1.16 IVolumeClient::FTReplaceMirrorPartition (Opnum 19)....................102
3.2.4.4.1.17 IVolumeClient::FTReplaceParityStripePartition (Opnum 20)............104
3.2.4.4.1.18 IVolumeClient::EnumDriveLetters (Opnum 21)............................106
3.2.4.4.1.19 IVolumeClient::AssignDriveLetter (Opnum 22)............................107
3.2.4.4.1.20 IVolumeClient::FreeDriveLetter (Opnum 23)...............................109
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)..................................114
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)..124
3.2.4.4.1.33 IVolumeClient::RemoveMirror (Opnum 37)..................................126
3.2.4.4.1.34 IVolumeClient::SplitMirror (Opnum 38).......................................128
3.2.4.4.1.35 IVolumeClient::InitializeDisk (Opnum 39)....................................129
3.2.4.4.1.36 IVolumeClient::UninitializeDisk (Opnum 40)................................131
3.2.4.4.1.37 IVolumeClient::ReConnectDisk (Opnum 41)................................132
3.2.4.4.1.38 IVolumeClient::ImportDiskGroup (Opnum 43).............................133
3.2.4.4.1.39 IVolumeClient::DiskMergeQuery (Opnum 44)..............................135
3.2.4.4.1.40 IVolumeClient::DiskMerge (Opnum 45).......................................136
3.2.4.4.1.41 IVolumeClient::ReAttachDisk (Opnum 47)...................................138
3.2.4.4.1.42 IVolumeClient::ReplaceRaid5Column (Opnum 51).........................139
3.2.4.4.1.43 IVolumeClient::RestartVolume (Opnum 52).................................141
3.2.4.4.1.44 IVolumeClient::GetEncapsulateDiskInfo (Opnum 53).....................142
3.2.4.4.1.45 IVolumeClient::EncapsulateDisk (Opnum 54)...............................146
3.2.4.4.1.46 IVolumeClient::QueryChangePartitionNumbers (Opnum 55)...........149
3.2.4.4.1.47 IVolumeClient::DeletePartitionNumberInfoFromRegistry (Opnum 56)
....150
3.2.4.4.1.48 IVolumeClient::SetDontShow (Opnum 57)...................................150
3.2.4.4.1.49 IVolumeClient::GetDontShow (Opnum 58)...................................151
3.2.4.4.1.50 IVolumeClient::EnumTasks (Opnum 67)......................................152
3.2.4.4.1.51 IVolumeClient::GetTaskDetail (Opnum 68).................................152
3.2.4.4.1.52 IVolumeClient::AbortTask (Opnum 69)..153
3.2.4.4.1.53 IVolumeClient::HrGetErrorData (Opnum 70)................................154
3.2.4.4.1.54 IVolumeClient::Initialize (Opnum 71)...155
3.2.4.4.1.55 IVolumeClient::Uninitialize (Opnum 72)......................................157
3.2.4.4.1.56 IVolumeClient::Refresh (Opnum 73)...157
3.2.4.4.1.57 IVolumeClient::RescanDisks (Opnum 74).....................................158
3.2.4.4.1.58 IVolumeClient::RefreshFileSys (Opnum 75).................................158
3.2.4.4.1.59 IVolumeClient::SecureSystemPartition (Opnum 76)......................158
3.2.4.4.1.60 IVolumeClient::ShutDownSystem (Opnum 77).............................159
3.2.4.4.1.61 IVolumeClient::EnumAccessPath (Opnum 78).............................159
3.2.4.4.1.62 IVolumeClient::EnumAccessPathForVolume (Opnum 79)...............160
3.2.4.4.1.63 IVolumeClient::AddAccessPath (Opnum 80)................................161
3.2.4.4.1.64 IVolumeClient::DeleteAccessPath (Opnum 81).............................161
3.2.4.4.2 IVolumeClient2..162
3.2.4.4.2.1 IVolumeClient2::GetMaxAdjustedFreeSpace (Opnum 3).................163
3.2.4.4.3 IVolumeClient3..163
3.2.4.4.3.1 IVolumeClient3::EnumDisksEx (Opnum 3)...................................166
3.2.4.4.3.2 IVolumeClient3::EnumDiskRegionsEx (Opnum 4)..........................167
3.2.4.4.3.3 IVolumeClient3::CreatePartition (Opnum 5)................................168
3.2.4.4.3.4 IVolumeClient3::CreatePartitionAssignAndFormat (Opnum 6).........168
3.2.4.4.3.5 IVolumeClient3::CreatePartitionAssignAndFormatEx (Opnum 7).....169

Copyright © 2016 Microsoft Corporation

3.2.4.4.3.6 IVolumeClient3::DeletePartition (Opnum 8) 170
3.2.4.4.3.7 IVolumeClient3::InitializeDiskStyle (Opnum 9) 170
3.2.4.4.3.8 IVolumeClient3::MarkActivePartition (Opnum 10) 171
3.2.4.4.3.9 IVolumeClient3::Eject (Opnum 11) 172
3.2.4.4.3.10 IVolumeClient3::FTEnumVolumes (Opnum 13) 172
3.2.4.4.3.11 IVolumeClient3::FTEnumLogicalDiskMembers (Opnum 14) 173
3.2.4.4.3.12 IVolumeClient3::FTDeleteVolume (Opnum 15) 173
3.2.4.4.3.13 IVolumeClient3::FTBreakMirror (Opnum 16) 174
3.2.4.4.3.14 IVolumeClient3::FTResyncMirror (Opnum 17) 174
3.2.4.4.3.15 IVolumeClient3::FTRegenerateParityStripe (Opnum 18) 175
3.2.4.4.3.16 IVolumeClient3::FTReplaceMirrorPartition (Opnum 19) 175
3.2.4.4.3.17 IVolumeClient3::FTReplaceParityStripePartition (Opnum 20) 176
3.2.4.4.3.18 IVolumeClient3::EnumDriveLetters (Opnum 21) 177
3.2.4.4.3.19 IVolumeClient3::AssignDriveLetter (Opnum 22) 177
3.2.4.4.3.20 IVolumeClient3::FreeDriveLetter (Opnum 23) 178
3.2.4.4.3.21 IVolumeClient3::EnumLocalFileSystems (Opnum 24) 179
3.2.4.4.3.22 IVolumeClient3::GetInstalledFileSystems (Opnum 25) 179
3.2.4.4.3.23 IVolumeClient3::Format (Opnum 26) 180
3.2.4.4.3.24 IVolumeClient3::EnumVolumes (Opnum 27) 180
3.2.4.4.3.25 IVolumeClient3::EnumVolumeMembers (Opnum 28) 181
3.2.4.4.3.26 IVolumeClient3::CreateVolume (Opnum 29) 181
3.2.4.4.3.27 IVolumeClient3::CreateVolumeAssignAndFormat (Opnum 30) 182
3.2.4.4.3.28 IVolumeClient3::CreateVolumeAssignAndFormatEx (Opnum 31) 182
3.2.4.4.3.29 IVolumeClient3::GetVolumeMountName (Opnum 32) 184
3.2.4.4.3.30 IVolumeClient3::GrowVolume (Opnum 33) 184
3.2.4.4.3.31 IVolumeClient3::DeleteVolume (Opnum 34) 185
3.2.4.4.3.32 IVolumeClient3::CreatePartitionsForVolume (Opnum 35) 185
3.2.4.4.3.33 IVolumeClient3::DeletePartitionsForVolume (Opnum 36) 187
3.2.4.4.3.34 IVolumeClient3::GetMaxAdjustedFreeSpace (Opnum 37) 188
3.2.4.4.3.35 IVolumeClient3::AddMirror (Opnum 38) 189
3.2.4.4.3.36 IVolumeClient3::RemoveMirror (Opnum 39) 189
3.2.4.4.3.37 IVolumeClient3::SplitMirror (Opnum 40) 190
3.2.4.4.3.38 IVolumeClient3::InitializeDiskEx (Opnum 41) 190
3.2.4.4.3.39 IVolumeClient3::UninitializeDisk (Opnum 42) 192
3.2.4.4.3.40 IVolumeClient3::ReConnectDisk (Opnum 43) 192
3.2.4.4.3.41 IVolumeClient3::ImportDiskGroup (Opnum 44) 192
3.2.4.4.3.42 IVolumeClient3::DiskMergeQuery (Opnum 45) 193
3.2.4.4.3.43 IVolumeClient3::DiskMerge (Opnum 46) 194
3.2.4.4.3.44 IVolumeClient3::ReAttachDisk (Opnum 47) 194
3.2.4.4.3.45 IVolumeClient3::ReplaceRaid5Column (Opnum 48) 195
3.2.4.4.3.46 IVolumeClient3::RestartVolume (Opnum 49) 195
3.2.4.4.3.47 IVolumeClient3::GetEncapsulateDiskInfoEx (Opnum 50) 196
3.2.4.4.3.48 IVolumeClient3::EncapsulateDiskEx (Opnum 51) 199
3.2.4.4.3.49 IVolumeClient3::QueryChangePartitionNumbers (Opnum 52) 202
3.2.4.4.3.50 IVolumeClient3::DeletePartitionNumberInfoFromRegistry (Opnum 53) 203
3.2.4.4.3.51 IVolumeClient3::SetDontShow (Opnum 54) 203
3.2.4.4.3.52 IVolumeClient3::GetDontShow (Opnum 55) 203
3.2.4.4.3.53 IVolumeClient3::EnumTasks (Opnum 64) 204
3.2.4.4.3.54 IVolumeClient3::GetTaskDetail (Opnum 65) 204
3.2.4.4.3.55 IVolumeClient3::AbortTask (Opnum 66) 205
3.2.4.4.3.56 IVolumeClient3::HrGetErrorData (Opnum 67) 205
3.2.4.4.3.57 IVolumeClient3::Initialize (Opnum 68) 206
3.2.4.4.3.58 IVolumeClient3::Uninitialize (Opnum 69) 207
3.2.4.4.3.59 IVolumeClient3::Refresh (Opnum 70) 207
3.2.4.4.3.60 IVolumeClient3::RescanDisks (Opnum 71) 207
3.2.4.4.3.61 IVolumeClient3::RefreshFileSys (Opnum 72) 208
3.2.4.4.3.62 IVolumeClient3::SecureSystemPartition (Opnum 73) 208

Copyright © 2016 Microsoft Corporation

3.2.4.4.3.63 IVolumeClient3::ShutDownSystem (Opnum 74) 208
3.2.4.4.3.64 IVolumeClient3::EnumAccessPath (Opnum 75)....... 208
3.2.4.4.3.65 IVolumeClient3::EnumAccessPathForVolume (Opnum 76)....... 209
3.2.4.4.3.66 IVolumeClient3::AddAccessPath (Opnum 77) 209
3.2.4.4.3.67 IVolumeClient3::DeleteAccessPath (Opnum 78)....... 210
3.2.4.4.4 IVolumeClient4 210
3.2.4.4.4.1 IVolumeClient4::RefreshEx (Opnum 3) 210
3.2.4.4.4.2 IVolumeClient4::GetVolumeDeviceName (Opnum 4) 211
3.2.4.4.5 IDMRemoteServer 212
3.2.4.4.5.1 IDMRemoteServer::CreateRemoteObject (Opnum 3)....... 212
3.2.5 Timer Events....... 212
3.2.6 Other Local Events....... 212
3.2.6.1 Disk Arrival 213
3.2.6.2 Disk Removal 213
3.2.6.3 Disk Layout Change 213
3.2.6.4 File System Change....... 213
3.2.6.5 Drive Letter Arrival....... 213
3.2.6.6 Drive Letter Removal....... 214
3.2.6.7 Media Arrival 214
3.2.6.8 Media Removal 214

4 Protocol Examples 215
4.1 Starting a New Session on a Local or Remote Server 215
4.2 Starting a New Session on a Remote Server Using the IDMRemoteServer Interface. 216
4.3 Creating a Partition 217
4.4 Deleting a Partition....... 219
4.5 Creating a Volume....... 221
4.6 Deleting a Volume....... 223

5 Security Considerations 226

6 Appendix A: Full IDL 227
6.1 Appendix A.1: dmintf.idl 227
6.2 Appendix A.2: dmintf3.idl....... 238

7 Appendix B: Product Behavior 246

8 Appendix C: IDMNotify::ObjectsChanged 259

9 Change Tracking 265

10 Index 266

Copyright © 2016 Microsoft Corporation

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 is assumed 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.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative~~ and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in [RFC2119]. Sections 1.5 and 1.9 are also normative but do not contain those terms~~. All other sections and examples in this specification are informative.

1.1 Glossary

~~The~~This document uses the following terms~~ are specific to this document~~:

active partition: A **partition** on a **master boot record (MBR) disk** that becomes the **system partition** at system startup if the basic input/output system (BIOS) is configured to select that **disk** for startup. An **MBR disk** can have exactly one active partition. The active partition is stored in the **partition table** on the **disk**. **GUID partitioning table (GPT) disks** do not have active partitions. See also **master boot record (MBR)**, **system partition**, and **partition table**.

allocation unit size: The size (expressed in bytes) of the units used by the **file system** to allocate space on a disk for the **file system** used by the **volume**. The size, in bytes, must be a power of two and must be a multiple of the size of the sectors on the disk. Typical **allocation unit sizes** of most **file systems** range from 512 bytes to 64 KB.

ASCII: The American Standard Code for Information Interchange (ASCII) is an 8-bit character-encoding scheme based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII refers to a single 8-bit ASCII character or an array of 8-bit ASCII characters with the high bit of each character set to zero.

basic disk: A disk on which each **volume** can be composed of exclusively one **partition**.

basic volume: A **partition** on a **basic disk**.

boot file: A file that contains a list of paths to **boot partitions**. On some systems, the **boot file** ~~may be~~is stored on other ~~non-volatile~~nonvolatile media, such as nonvolatile random access memory (NVRAM~~.~~)..

boot loader: An architecture-specific file that loads the operating system on the **boot partition** as specified by the boot configuration file.

boot loader file: See **boot loader**.

boot partition: A **partition** containing the operating system.

boot volume: See **boot partition**.

boot.ini: The name of the **boot loader file** on Windows–based computers.

Copyright © 2016 Microsoft Corporation

bus: Computer hardware to which peripheral devices ~~may~~can be connected. Messages are sent between the CPU and the peripheral devices using the **bus**. Examples of **bus** types include SCSI, **USB**, and 1394.

bus type: A type of **bus**. Examples of **bus types** include SCSI, **USB**, and 1394.

Compact Disc File System (CDFS): A file system used for storing files on CD-ROMs.

Component Object Model (COM): An object-oriented programming model that defines how objects interact within a single process or between processes. In **COM**, clients have access to an object through interfaces implemented on the object. For more information, see [MS-DCOM].

crash dump file: A file that ~~may~~can be created by an operating system when an unrecoverable fault occurs. This file contains the contents of memory at the time of the crash and ~~may~~can be used to debug the problem ~~.~~creator.

cylinder: The set of disk tracks that appear in the same location on each platter of a disk.

disk: A persistent storage device that can include physical hard disks, removable disk units, optical drive units, and logical unit numbers (LUNs) unmasked to the system.

disk adapter: Computer hardware that controls a disk.

disk encapsulation: The process of converting a **basic disk** to a **dynamic disk**. Encapsulating a disk lays down disk metadata that is used for managing the disk dynamically.

disk extent: A contiguous set of one or more disk sectors. A disk extent can be used as a partition or part of a volume, or it can be free, which indicates that it is not in use or that it ~~may~~might be unusable for creating partitions or volumes.

disk group: In the context of **dynamic disks**, this term describes a logical grouping of disks.

disk group import: The act of merging a set of disks belonging to one disk group into another set of disks belonging to a second disk group. The result is a single disk group that includes all disks involved in the import.

disk regions: See **disk extent**.

disk signature: A unique identifier for a disk. For a **master boot record (MBR)**-formatted disk, this identifier is a 4-byte value stored at the end of the **MBR**, which is located in sector 0 on the disk. For a **GUID partitioning table (GPT)**-formatted disk, this value is a **GUID** stored in the **GPT** disk header at the beginning of the disk.

disk type: A disk that is hardware-specific. A disk can only communicate with the CPU using a bus of matching type. Examples of bus types include SCSI, USB, and 1394.

Distributed Component Object Model (DCOM): The Microsoft Component Object Model (COM) specification that defines how components communicate over networks, as specified in [MS-DCOM].

drive letter: One of the 26 alphabetical characters A-Z, in uppercase or lowercase, that is assigned to a volume. Drive letters serve as a namespace through which data on the volume can be accessed. A volume with a drive letter can be referred to with the drive letter followed by a colon (for example, C:).

dynamic disk: A disk on which volumes ~~may~~can be composed of more than one partition on disks of the same pack, as opposed to basic disks where a partition and a volume are equivalent.

dynamic volume: A volume on a dynamic disk.

Copyright © 2016 Microsoft Corporation

extended partition: A construct that is used to partition a disk into logical units. A disk ~~may~~can have up to four primary **partitions** or up to three primary **partitions** and one extended **partition**. The extended **partition** ~~may~~can be further subdivided into multiple logical drives.

extent: A contiguous area of storage in a computer file system, reserved for a file.

FAT file system: A **file system** used by MS-DOS and other Windows operating systems to organize and manage files. The **file allocation table (FAT)** is a data structure that the operating system creates when a **volume** is formatted by using **FAT** or **FAT32 file systems**. The operating system stores information about each file in the **FAT** so that it can retrieve the file later.

FAT32 file system: A derivative of the **file allocation table (FAT)** file system. **FAT32** supports smaller cluster sizes and larger **volumes** than **FAT**, which results in more efficient space allocation on **FAT32 volumes**. **FAT32** uses 32-bit addressing.

fault-tolerant: The ability of computer hardware or software to ensure data integrity when hardware failures occur. Fault-tolerant features appear in many server operating systems and include mirrored volumes and RAID-5 volumes. A fault-tolerant volume maintains more than one copy of the volume's data. In the event of disk failure, a copy of the data is still available.

fault-tolerant mirror set: A **volume** configuration such that more than one copy of the **volume data** is maintained. Each copy of the data is placed on separate sets of disks. If a disk in one disk set fails, the **volume's** data is still available on the second set of disks.

file allocation table (FAT): A data structure that the operating system creates when a volume is formatted by using **FAT** or **FAT32 file systems**. The operating system stores information about each file in the **FAT** so that it can retrieve the file later.

file allocation units: Units of a specific size that are used by the **file system** to allocate space on a disk for the **file system** used by the **volume**.

file system: A system that enables applications to store and retrieve files on storage devices. Files are placed in a hierarchical structure. The file system specifies naming conventions for files and the format for specifying the path to a file in the tree structure. Each file system consists of one or more drivers and DLLs that define the data formats and features of the file system. File systems can exist on the following storage devices: diskettes, hard disks, jukeboxes, removable optical disks, and tape backup units.

file system flags: A set of values used by a **file system** to configure and report **file system** features and operations.

flags: A set of values used to configure or report options or settings.

foreign: A dynamic disk group that is not part of a machine's primary disk group. The term **foreign** denotes "foreign to this machine". **Foreign** disk and **foreign** disk groups are not online. This means that these disks may not be configured and no data input/output (I/O) to the disks or the **volumes** on the disks is permitted.

format: To submit a command for a **volume** to write metadata to the disk, which is used by the **file system** to organize the data on the disk. A volume is **formatted** with a specific **file system**.

free space: Space on a disk not in use by any **volumes**, primary partitions, or logical drives.

full format: A format in which all data sectors for the **volume** are initialized at the time the **file system** metadata is created.

globally unique identifier (GUID): A term used interchangeably with **universally unique identifier (UUID)** in Microsoft protocol technical documents (TDs). Interchanging the usage of

Copyright © 2016 Microsoft Corporation

these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also **universally unique identifier (UUID)**.

GUID partition table (GPT): A disk-partitioning scheme that is used by the Extensible Firmware Interface (EFI). **GPT** offers more advantages than **master boot record (MBR)** partitioning because it allows up to 128 **partitions** per disk, provides support for **volumes** up to 18 exabytes in size, allows primary and backup **partition tables** for redundancy, and supports unique disk and partition IDs through the use of **globally unique identifiers (GUIDs)**. Disks with **GPT** schemes are referred to as **GPT** disks.

hard disk physical name: An implementation-specific **path** that can be used to refer to a specific hard disk on a machine.

hibernation image: An image that contains metadata required to support a Windows operating system feature known as hibernation. Hibernation allows a system's state to be preserved in persistent storage while the system is shut down.

Integrated Drive Electronics (IDE) bus: A standard electronic interface used between a computer motherboard's **bus** and the computer's disk storage devices.

locked partition: A partition that is inaccessible.

Logical Disk Manager (LDM): A subsystem of Windows that manages dynamic disks. Dynamic disks contain a master boot record (MBR) at the beginning of the disk, one **LDM** partition, and an **LDM** database at the end. The **LDM** database contains partitioning information used by the **LDM**.

logical drive: A set of disk **extents** that compose a **volume**.

logical partition: See **logical drive**.

mass storage device: Any hardware device that provides persistent storage of data.

master boot record (MBR): Metadata such as the partition table, the disk signature, and the executable code for initiating the operating system boot process that is located on the first sector of a disk. Disks that have **MBRs** are referred to as **MBR** disks. **GUID partitioning table (GPT)** disks, instead, have unused dummy data in the first sector where the **MBR** would normally be.

Microsoft Interface Definition Language (MIDL): The Microsoft implementation and extension of the OSF-DCE Interface Definition Language (IDL). **MIDL** can also mean the Interface Definition Language (IDL) compiler provided by Microsoft. For more information, see [MS-RPCE].

mirrored volume: A fault-tolerant volume that maintains two or more copies of the volume's data. In the event that a disk is lost, at least one copy of the volume's data remains and can be accessed.

modification sequence number: An implementation-defined value for objects such as disks, volumes, drive letters, partitions, and regions that increases monotonically each time a configuration operation takes place on the object.

mount path: See **mounted folder**.

mount point: See **mounted folder**.

mounted folder: A file system directory that contains a linked path to a second volume. A user ~~may~~can link a path on one volume to another. For example, given two volumes C: and D:, a user can create a directory or folder C:\mountD and link that directory with volume D:. The path C:\MountD can then be used to access the root folder of volume D:.

Copyright © 2016 Microsoft Corporation

NT file system (NTFS): ~~**NT file system (NTFS)** is a~~A proprietary Microsoft ~~File System~~file system. For more information, see [MSFT-NTFS].

object identifier (OID): In the context of an object server, a 64-bit number that uniquely identifies an object.

online: An operational state applicable to **volumes** and disks. In the online state, the volume or disk is available for data input/output (I/O) or configuration.

page file or paging file: A file that is used by operating systems for managing virtual memory.

partition: In the context of hard disks, a logical region of a hard disk. A hard disk may be subdivided into one or more **partitions**.

partition table: An area of a disk that is used to store metadata information about the **partitions** on the disk. See also, **GUID partitioning table (GPT)**.

partition type: A value indicating the **partition's** intended use, or indicating the type of file system on the **partition**. For example, **partition** type 0x07 indicates that the **partition** is formatted with the NTFS file system. Original equipment manufacturers ~~may~~can designate a **partition** type of 0x12 to indicate that manufacturer-specific data is stored on the **partition**.

path: When referring to a file path on a file system, a hierarchical sequence of folders. When referring to a connection to a storage device, a connection through which a machine can communicate with the storage device.

primary disk group: In the context of **dynamic disk**, it is the disk group whose disks are online, which means they are accessible for input/output (I/O) and configuration. Each machine ~~may~~can have only one primary disk group. Disks on the machine belonging to other disk groups are referred to as "foreign disks" and their disk group is referred to as a "foreign disk group".

primary partition: A type of partition on a **master boot record (MBR)**-formatted disk.

quick format: A formatting that does not zero the data sectors on the volume at the time the file system metadata is created.

RAID-5: A fault-tolerant volume that maintains the volume's data across multiple RAID columns. Fault tolerance is provided by writing parity data for each stripe. In the event that one disk encounters a fault, that disk's data ~~may~~can be reconstructed using the parity data located on the other disks.

redundant arrays of independent disks (RAID): A set of disk-organization techniques that is designed to achieve high-performance storage access and availability.

region: See **disk extent**.

region flags: A set of values that describes the region's state or use.

region's status: The status of the region, such as whether the region is performing properly or encountering disk faults.

remote procedure call (RPC): A context-dependent term commonly overloaded with three meanings. Note that much of the industry literature concerning RPC technologies uses this term interchangeably for any of the three meanings. Following are the three definitions: (*) The runtime environment providing remote procedure call facilities. The preferred usage for this meaning is "RPC runtime". (*) The pattern of request and response message exchange between two parties (typically, a client and a server). The preferred usage for this meaning is "RPC exchange". (*) A single message from an exchange as defined in the previous definition. The preferred usage for this term is "RPC message". For more information about RPC, see [C706].

Copyright © 2016 Microsoft Corporation

removable media: Any type of storage that is not permanently attached to the computer. A persistent storage device stores its data on media. If the media can be removed from the device, the media is considered removable. For example, a floppy disk drive uses removable media.

RPC protocol sequence: A character string that represents a valid combination of a **remote procedure call (RPC)** protocol, a network layer protocol, and a transport layer protocol, as described in [C706] and [MS-RPCE].

SCSI logical unit number (LUN): See logical unit number (LUN).

SCSI port number: A number that uniquely identifies a port on a small computer system interface (SCSI) disk controller. Each SCSI disk controller ~~may~~can support multiple SCSI bus attachments or ports for connecting SCSI devices to a computer.

sector: The smallest addressable unit of a disk.

serial storage architecture (SSA) bus: Serial storage architecture (SSA) is a standard for high-speed access to high-capacity disk storage. An **SSA bus** is implemented to the SSA standard.

simple volume: A **volume** whose data exists on a single **partition**.

small computer system interface (SCSI) bus: A standard for connecting peripheral devices to a computer. A **SCSI bus** is an implementation of this standard.

system directory: A directory that contains system files comprising the operating system.

system partition: A partition that contains the boot loader needed to invoke the operating system on the boot partition. A system partition must also be an active partition. It can be, but is not required to be, the same partition as the boot partition.

track: Any of the concentric circles on a disk platter over which a magnetic head (used for reading and writing data on the disk) passes while the head is stationary but the disk is spinning. A track is subdivided into sectors, upon which data is read and written.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The **Unicode** standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

unique identifier (UID): A pair consisting of a **GUID** and a version sequence number to identify each resource uniquely. The UID is used to track the object for its entire lifetime through any number of times that the object is modified or renamed.

Universal Disk Format (UDF): A type of file system for storing files on optical media.

universal serial bus (USB): An external bus that supports Plug and Play installation. It allows devices to be connected and disconnected without shutting down or restarting the computer.

universally unique identifier (UUID): A 128-bit value. UUIDs can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects in cross-process communication such as client and server interfaces, manager entry-point vectors, and **RPC** objects. UUIDs are highly likely to be unique. UUIDs are also known as **globally unique identifiers (GUIDs)** and these terms are used interchangeably in the Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the UUID. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the UUID.

Copyright © 2016 Microsoft Corporation

user-defined function (UDF): A function that is coded in a Microsoft Visual Basic for Applications (VBA) module, macro sheet, add-in, or Excel Linked Library (XLL). A UDF can be used in formulas to return values to a worksheet, similar to built-in functions.

volume: A group of one or more partitions that forms a logical region of storage and the basis for a file system. A **volume** is an area on a storage device that is managed by the file system as a discrete logical storage unit. A partition contains at least one **volume**, and a volume can exist on one or more partitions.

volume data: Data stored on a **volume**.

volume label: See file system label.

volume manager: A system component that manages communication and data transfer between applications and disks.

volume members: See RAID column.

volume plex: A member of a **volume** that represents a complete copy of data stored. For instance, mirrored volumes have more than one plex.

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

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

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.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, https://www2.opengroup.org/ogsys/catalog/c706

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

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

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

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

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

1.2.2 Informative References

[MS-VDS] Microsoft Corporation, "Virtual Disk Service (VDS) Protocol".

[MSDN-AccPerms] Microsoft Corporation, "AccessPermission", http://msdn.microsoft.com/en-us/library/ms688679.aspx

Copyright © 2016 Microsoft Corporation

[MSDN-DefAccPerms] Microsoft Corporation, "DefaultAccessPermission", http://msdn.microsoft.com/en-us/library/ms678417(VS.85).aspx

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

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

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

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

1.3 Overview

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 implements supports that let the DCOM interface handle requests for storage management services for a server system over the network. 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. The IVolumeClient and IVolumeClient2 interfaces do not support the GPT disk-partitioning style and ~~MUST not~~cannot be used with GPT partitioned disks.

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

IDMRemoteServer includes functionality to create an instance of the Disk Management server on a remote machine.

IDMNotify is the interface implemented by the client to receive notifications from the Disk Management server.<1>

Copyright © 2016 Microsoft Corporation

1.4 Relationship to Other Protocols

The Disk Management Remote Protocol relies on the Distributed Component Object Model (DCOM) Remote Protocol (as specified in [MS-DCOM]), which uses **remote procedure call (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 can 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 IDMRemoteServer	3A410F21-553F-11d1-8E5E-00A0C92C9D5D	Section 2.1
RPC interface UUID for IDMNotify	D2D79DF7-3400-11d0-B40B-00AA005FF586	Section 2.1

Copyright © 2016 Microsoft Corporation

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]), 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] section 2.1, 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
0xC1000001 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.
0xC1000004 LDM_E_LOST	INTERNAL Error — The commit status has been lost.
0xC100000A LDM_E_BADMSG	INTERNAL Error — The request contains a message format that is not valid.

Copyright © 2016 Microsoft Corporation

Return value/code	Description
0xC100000C LDM_E_NOMEM	There is not enough memory available to finish the operation. Save the work, exit other programs, and then try again.
0xC1000014 LDM_E_NOTRANS	INTERNAL Error — The operation requires a transaction.
0xC1000016 LDM_E_RESTART	INTERNAL Error — The commit has been aborted, transaction must be restarted.
0xC1000017 LDM_E_DB_RESTART	The dynamic disks present in the system cannot be configured due to excessive disk input/output (I/O) errors. Verify the status of storage devices, and then restart the computer.
0xC1000018 LDM_E_VOLDERR	INTERNAL Error — An unexpected error occurred in the configuration program.
0xC1000019 LDM_E_REPLAY	INTERNAL Error — An unexpected error occurred in the kernel during a configuration update.
0xC100001B LDM_E_LOGIO_FAIL	INTERNAL Error — **Logical Disk Manager (LDM)** cannot start the logging thread.
0xC100001C LDM_E_DG_DISABLED	The **disk group** has been disabled.
0xC1000029 LDM_E_DIAG_DBASE	One or more errors exist in the disk group configuration copies.
0xC100002D LDM_E_DIAG_DISABLED	INTERNAL Error — The configuration program is currently disabled.
0xC1000030 LDM_E_DIAG_CYCLE	INTERNAL Error — No convergence between the disk group and disk list.
0xC1000032 LDM_E_VB_NOENT	The expected LDM registry keys were not found.
0xC1000033 LDM_E_VB_SYS	A system error was encountered while reading or setting an LDM registry key.
0xC1000034 LDM_E_VB_FORMAT	INTERNAL Error — String format errors were found in LDM registry keys.
0xC100003C LDM_E_INPROGRESS	INTERNAL Error — A transaction is already in progress.
0xC100003D LDM_E_DISABLED	INTERNAL Error — This feature is disabled or is not implemented.
0xC100003E LDM_E_NOSUPPORT	INTERNAL Error — The requested operation is not supported.
0xC100003F LDM_E_LOCK	INTERNAL Error — The requested lock could not be obtained.
0xC1000040	INTERNAL Error — The required lock is not held in

Copyright © 2016 Microsoft Corporation

Return value/code	Description
LDM_E_NOLOCK	the transaction.
0xC1000041 LDM_E_NO_DLOCK	INTERNAL Error — The required data lock is not held in the transaction.
0xC1000042 LDM_E_EXIST	INTERNAL Error — An object with the specified name already exists in the disk group.
0xC1000043 LDM_E_NOENT	The specified object no longer exists in the disk group.
0xC1000044 LDM_E_CONFIG	INTERNAL Error — The disk group configuration has changed.
0xC1000046 LDM_E_BUSY	The specified object is in active use; operation not allowed.
0xC1000047 LDM_E_INVAL	The requested operation is not valid.
0xC1000049 LDM_E_NEG_SIZE	INTERNAL Error — Negative length, width, or offset received as parameter of the request.
0xC100004B LDM_E_BADNAME	INTERNAL Error — The record name is not valid.
0xC100004C LDM_E_NOT_DIS	The operation requires a disabled volume.
0xC100004D 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.
0xC100004E LDM_E_NO_ASSOC	INTERNAL Error — The operation requires an associated record.
0xC100004F LDM_E_ASSOC	The operation is not allowed because the requested object is in active use.
0xC1000052 LDM_E_HOLE	INTERNAL Error — The specified striped plex is not compact.
0xC1000053 LDM_E_DIFF_SIZE	INTERNAL Error — The subdisks of the striped plex have different sizes.
0xC1000054 LDM_E_NONE_ASSOC	INTERNAL Error — The striped plex has no associated subdisks.
0xC1000055 LDM_E_FULL_ASSOC	INTERNAL Error — The record cannot have more associations.
0xC1000056 LDM_E_INVAL_RTYPE	INTERNAL Error — The record type is not valid.
0xC1000057 LDM_E_NOT_SUBDISK	INTERNAL Error — The specified record is not a subdisk.

Copyright © 2016 Microsoft Corporation

Return value/code	Description
0xC1000058 LDM_E_NOT_PLEX	INTERNAL Error — The specified record is not a plex.
0xC1000059 LDM_E_NOT_VOL	INTERNAL Error — The specified record is not a volume.
0xC100005A LDM_E_NOT_DM	INTERNAL Error — The specified record is not a disk media.
0xC100005B LDM_E_NOT_DG	INTERNAL Error — The specified record is not a disk group.
0xC100005C LDM_E_NOT_DA	INTERNAL Error — The specified record is not a disk access.
0xC100005D LDM_E_IS_DG	INTERNAL Error — The operation is not allowed on a disk group.
0xC100005E LDM_E_MAX_VOL	The volume limit has been reached; no additional **dynamic volumes** can be created.
0xC100005F LDM_E_MAX_PLEX	INTERNAL Error — Too many plexes exist.
0xC1000060 LDM_E_OVERLAP	INTERNAL Error — The subdisk overlaps with another subdisk.
0xC1000061 LDM_E_TOO_SMALL	INTERNAL Error — A striped plex subdisk has the length shorter than the stripe width.
0xC1000062 LDM_E_SD_WIDTH	INTERNAL Error — A striped plex subdisk has the length not multiple of the stripe width.
0xC1000063 LDM_E_INVAL_FIELD	INTERNAL Error — The specified field is not valid.
0xC1000064 LDM_E_ST_WIDTH	INTERNAL Error — The plex stripe width is not valid.
0xC1000065 LDM_E_OVERFLOW	INTERNAL Error — The operation overflows the maximum offsets.
0xC1000066 LDM_E_LOG_SD_SMALL	INTERNAL Error — The log subdisk is too small for the volume.
0xC1000069 LDM_E_HAS_LOG	INTERNAL Error — The plex already has a log subdisk.
0xC100006B LDM_E_NO_DG	The specified disk group no longer exists.
0xC100006C LDM_E_LDM_E_DG_MISMATCH	INTERNAL Error — The request crosses a disk group boundary.
0xC100006D LDM_E_NOT_ROOTDG	INTERNAL Error — The request is allowed only in the root disk group.

Copyright © 2016 Microsoft Corporation

Return value/code	Description
0xC100006E LDM_E_PUB_BOUNDS	INTERNAL Error — The subdisk is not within the public region boundaries.
0xC100006F LDM_E_BAD_DISK	The specified disk is not ready or usable.
0xC1000070 LDM_E_VOL_INVALID	INTERNAL Error — The specified volume is not usable.
0xC1000071 LDM_E_PATH_STAT	The specified device path cannot be accessed.
0xC1000073 LDM_E_NOROOTDG	INTERNAL Error — The root disk group is not enabled.
0xC1000074 LDM_E_NODATATYPE	INTERNAL Error — The disk access type is not recognized.
0xC1000075 LDM_E_DAOFFLINE	The specified disk is offline.
0xC1000076 LDM_E_REC_TOOBIG	INTERNAL Error — The record store size is too large.
0xC1000077 LDM_E_SD_BAD	INTERNAL Error — The specified subdisk is not enabled.
0xC1000078 LDM_E_PLEX_BAD	INTERNAL Error — The specified plex contains disabled subdisks.
0xC1000079 LDM_E_NOT_EMPTY	The operation is not allowed because the specified disk is not empty.
0xC100007A LDM_E_DG_EXIST	INTERNAL Error — A disk group with the specified name already exists.
0xC100007B LDM_E_DGCREATED	INTERNAL Error — The operation is not allowed on a created disk group.
0xC100007C LDM_E_DG_LOG_FULL	INTERNAL Error — The configuration is too large for the disk group log.
0xC100007D LDM_E_DG_LOG_SMALL	INTERNAL Error — The disk log is too small for the disk group configuration.
0xC100007E LDM_E_DG_IMPORTED	INTERNAL Error — The disk group has already been imported.
0xC100007F LDM_E_DB_REC_ERROR	INTERNAL Error — An error exists in the configuration record.
0xC1000080 LDM_E_DB_TOO_BIG	INTERNAL Error — The configuration is too large to make configuration copies.
0xC1000081 LDM_E_DGNOTCREATED	INTERNAL Error — The disk group creation is not finish.

Copyright © 2016 Microsoft Corporation

Return value/code	Description
0xC1000082 LDM_E_LOG_BAD	INTERNAL Error — There are no valid log copies in the disk group.
0xC1000083 LDM_E_ROOTDG	INTERNAL Error — The operation is not allowed on the root disk group.
0xC1000084 LDM_E_LAST_DISK	INTERNAL Error — LDM cannot remove the last disk in the disk group.
0xC1000085 LDM_E_LAST_CONFIG	INTERNAL Error — LDM cannot remove the last disk group configuration copy.
0xC1000086 LDM_E_LAST_LOG	INTERNAL Error — LDM cannot remove the last disk group log copy.
0xC1000087 LDM_E_DB_FULL	The LDM configuration is full. No more objects can be created.
0xC1000088 LDM_E_DB_NOENT	INTERNAL Error — The database file cannot be found.
0xC1000089 LDM_E_DB_SYS	INTERNAL Error — System error in configuration copy.
0xC100008A LDM_E_DB_FORMAT	INTERNAL Error — A **format** error was found in the configuration copy.
0xC100008B LDM_E_DB_VERSION	The configuration format version is not supported.
0xC100008C LDM_E_DB_NOSPC	INTERNAL Error — The LDM configuration is full.
0xC100008D LDM_E_DB_SHORT	INTERNAL Error — Unexpected end of the configuration copy.
0xC100008E LDM_E_DB_READ	The LDM could not read the disk configuration.
0xC100008F LDM_E_DB_WRITE	The LDM could not write the disk configuration.
0xC1000090 LDM_E_DB_AS_BAD	INTERNAL Error — Association not resolved
0xC1000091 LDM_E_DB_ASCNT_BAD	INTERNAL Error — The association count is incorrect.
0xC1000094 LDM_E_DB_MAGIC	INTERNAL Error — Invalid magic number in the configuration copy.
0xC1000095 LDM_E_DB_BLKNO	INTERNAL Error — Invalid block number in the configuration copy.
0xC1000096 LDM_E_DB_NOMATCH	INTERNAL Error — No valid disk belonging to the disk group was found.

Copyright © 2016 Microsoft Corporation

Return value/code	Description
0xC1000097 LDM_E_DB_DUPLICATE	INTERNAL Error — Duplicate records exist in the configuration.
0xC1000098 LDM_E_DB_INCONSISTENT	INTERNAL Error — The configuration records are inconsistent.
0xC1000099 LDM_E_DB_NO_DGREC	INTERNAL Error — No disk group record exists in the configuration.
0xC100009A LDM_E_DB_BAD_TEMP	INTERNAL Error — The temporary and permanent configurations do not match.
0xC100009B LDM_E_DB_CHANGED	INTERNAL Error — The on-disk configuration changed during recovery.
0xC100009E LDM_E_KERN_KNOENT	INTERNAL Error — A record that was expected was not found in the kernel.
0xC100009F LDM_E_KERN_DBNOENT	INTERNAL Error — A record that exists in the kernel was not found in the configuration.
0xC10000A0 LDM_E_KERN_KDIFF	INTERNAL Error — The configuration record does not match the kernel.
0xC10000A1 LDM_E_KERN_INCONSISTENT	INTERNAL Error — The kernel and on-disk configurations do not match.
0xC10000A2 LDM_E_PUB_SIZE	INTERNAL Error — The public region of the disk is too small.
0xC10000A3 LDM_E_PRIV_SIZE	INTERNAL Error — The private region of the disk is too small.
0xC10000A4 LDM_E_PRIV_FULL	INTERNAL Error — The private region of the disk is full.
0xC10000A5 LDM_E_PRIV_FORMAT	INTERNAL Error — A format error was found in the private region of the disk.
0xC10000A6 LDM_E_PRIV_HEADMATCH	INTERNAL Error — The disk has inconsistent disk headers.
0xC10000A7 LDM_E_PRIV_NOHEADER	INTERNAL Error — The disk header cannot be found.
0xC10000A8 LDM_E_PRIV_INVAL	INTERNAL Error — The disk private region contents are not valid.
0xC10000A9 LDM_E_PRIV_VERSION	INTERNAL Error — The disk private region version is not supported.
0xC10000AA LDM_E_PRIV_INCONSISTENT	INTERNAL Error — The disks in the disk group are inconsistent.
0xC10000AB LDM_E_FIELD_NOREINIT	INTERNAL Error — The attribute cannot be changed by reinitializing the disk.

Copyright © 2016 Microsoft Corporation

Return value/code	Description
0xC10000AC LDM_E_DISK_INITED	INTERNAL Error — The disk has already been initialized.
0xC10000AD LDM_E_DISK_ALIASED	INTERNAL Error — The disk header indicates aliased partitions.
0xC10000AE LDM_E_DISK_SHARED	The disk is marked as shared. It cannot be used in the base Logical Disk Management product.
0xC10000AF LDM_E_DISK_WRONGHOST	The disk is marked as in use by another computer.
0xC10000B1 LDM_E_MISSING_PUB	INTERNAL Error — The disk has no public partition.
0xC10000B5 LDM_E_BAD_SCTR_SIZE	The disk **sector** size is not supported by the LDM.
0xC10000B6 LDM_E_NO_CONFIGS	INTERNAL Error — The disk group contains no valid configuration copies.
0xC10000B7 LDM_E_DISK_NOT_FOUND	The specified disk cannot be located.
0xC10000B8 LDM_E_DISK_OTHER_DG	INTERNAL Error — The disk belongs to another disk group.
0xC10000B9 LDM_E_BADCOLUMN	INTERNAL Error — The stripe column number is too large for the plex.
0xC10000BA LDM_E_NOTRAIDVOL	INTERNAL Error — The volume does not have **redundant arrays of independent disks (RAID)** read policy.
0xC10000BB LDM_E_RAIDVOL	INTERNAL Error — The volume has RAID read policy.
0xC10000BC LDM_E_TOOMANYRAID	INTERNAL Error — The volume already has one RAID plex.
0xC10000BE LDM_E_LIC	The license has expired or is not available for the operation.
0xC10000C0 LDM_E_NOTSTORAGE	INTERNAL Error — The volume does not have the storage attribute.
0xC10000C1 LDM_E_SUBVOLUME	INTERNAL Error — The subdisk is defined on a volume.
0xC10000C3 LDM_E_VOLTOOSMALL	INTERNAL Error — The volume length is too small to hold subdisks.
0xC10000C4 LDM_E_HASSUBVOLUME	INTERNAL Error — One or more subdisks are defined on the volume.
0xC10000C5 LDM_E_BADRAIDCHANGE	The operation is not allowed because it would make the **RAID-5** volume unusable.

Copyright © 2016 Microsoft Corporation

Return value/code	Description
0xC10000C6 LDM_E_BADRAID	The operation is not allowed because the RAID-5 volume is currently unusable.
0xC10000C7 LDM_E_PLEX_DIS	INTERNAL Error — The plex is disabled.
0xC10000CC LDM_E_MPTH_DISABLE	INTERNAL Error — The record is subsumed by a multipath disk.
0xC10000D1 LDM_E_NT_IF_ERROR	The operation was unsuccessful because a system error occurred.
0xC10000D2 LDM_E_TASK_ABORTED	The operation was canceled at the user's request.
0xC10000D3 LDM_E_TASK_IOERROR	The operation was canceled due to a disk I/O error.
0xC10000D4 LDM_E_TASK_DELETED	INTERNAL Error — The task has been deleted.
0xC10000D5 LDM_E_V_NOT_ENABLED	This operation requires a usable volume.
0xC10000D6 LDM_E_SPC_SETUP_FAIL	There is not enough free disk space to satisfy the request.
0xC10000D7 LDM_E_VOL_SETUP_FAIL	INTERNAL Error — The volume creation setup failed.
0xC10000D8 LDM_E_MIR_SETUP_FAIL	INTERNAL Error — The mirrored volume creation setup failed.
0xC10000DA LDM_E_GRW_SETUP_FAIL	INTERNAL Error — The volume grow operation setup failed.
0x410000DB LDM_E_FOREIGN_DA	The operation is not allowed because the disk contains **foreign** partitions.
0x410000DC LDM_E_PRIMARY_EXISTS	INTERNAL Error — A **primary disk group** already exists.
0x410000DD LDM_E_CAPS_VIOLATION	The requested operation is not allowed in Windows Professional.
0x410000DE LDM_E_VOL_NO_ZEROING	INTERNAL Error — The operation is not allowed because the volume is still initializing.
0x410000DF LDM_E_DISK_IN_USE	The requested disk is already in use by this volume.
0x410000E0 LDM_E_SDNODETACH	INTERNAL Error — A subdisk of the column is not detached.
0x410000E1 LDM_E_NOBADDISK	The operation is not needed because no mirrors or RAID-5 members are currently disabled.

Copyright © 2016 Microsoft Corporation

Return value/code	Description
0x410000E2 LDM_E_DEGRADED	The operation is not needed because no RAID-5 members are currently disabled.
0x410000E3 LDM_E_NO_OPEN	LDM could not open a volume.
0x410000E5 LDM_E_NO_DISMOUNT	LDM could not dismount a volume.
0x410000E6 LDM_E_BAD_PARTITION	LDM could not determine a **partition type**.
0x410000E7 LDM_E_MNT_FAIL	LDM could not associate a **drive letter**.
0x410000E8 LDM_E_FT_QUERY_FAIL	LDM could not retrieve information about a legacy basic volume.
0x410000E9 LDM_E_NO_FT	There is no legacy basic volume on the disk.
0x410000EA LDM_E_INSUFFICIENT_SPACE	There is not enough space for the configuration database on the disk.
0x410000EB LDM_E_FT_UNHEALTHY	A legacy basic volume is unhealthy.
0x410000ED LDM_E_NEEDS_REBOOT	The conversion will require restarting the computer.
0x410000EE LDM_E_BAD_ACTIVEPARTITION	An **active partition** is not the current active partition.
0x410000F0 LDM_E_TOO_MANY_PARTITIONS	The disk cannot be partitioned after it has been converted.
0x410000F1 LDM_E_ENCAP_PENDING	A disk conversion is already pending.
0x410000F2 LDM_E_ENCAP_FAIL	The LDM could not convert the selected disks.
0x410000F3 LDM_E_WRONG_DISKSET_ID	The disk configuration was changed on another system.
0x410000F4 LDM_REENCAP_ABORT	The LDM detected a previous conversion attempt failure.
0x410000F5 LDM_ENCAP_DONE	The selected disk has already been converted.
0x410000F6 LDM_FT_INCONSISTENT	The recorded configuration of legacy basic volumes before the conversion does not match the current configuration.
0x410000F7 LDM_MIXED_PARTITIONS	A foreign **partition** has been detected between basic partitions.

Copyright © 2016 Microsoft Corporation

Return value/code	Description
0xC10000F8 LDM_E_CARVE_PARTITION	The LDM could not create a partition for the specified volume.
0xC10000F9 LDM_E_FT_PRESENT	The LDM does not support conversion to dynamic for disks that contain legacy basic volumes.
0xC10003E8 LDM_E_NODGINFO	The LDM could not find the primary disk group.
0xC10003EA LDM_E_INVALIDOBJID	The type of a cached object is not valid.
0xC10003EB LDM_E_NORECORD	A cached object could not be found.
0xC10003ED LDM_E_NOTVMDISK	The disk does not belong to a disk group.
0xC10003F2 LDM_E_VOLUME_IN_USE	The request cannot be finished because the volume is open or in use.
0xC10003F3 LDM_E_FORMATINPROGRESS	The volume specified cannot be formatted because the system is busy formatting another volume. Wait until that format operation is finished before continuing.
0xC10003F5 LDM_E_OBJECT_STALED	The disk and volume information in the Disk Management snap-in is out of date. To refresh thedisk and volume information, press F5, or click Refresh on the Action menu.
0xC10003F6 LDM_E_BAD_FS	The operation did not finish because the file system is not compatible.
0xC10003F7 LDM_E_BAD_MEDIA	The operation did not finish because the media is not compatible.
0xC10003F8 LDM_E_NOACCESS	The operation did not finish because access is denied. Check the access permissions.
0xC10003F9 LDM_E_WRITE_PROTECTED	The operation did not finish because the media is write-protected.
0xC10003FA LDM_E_BAD_LABEL	The operation did not finish because the label supplied is not valid.
0xC10003FB LDM_E_CANNOT_QUICK_FORMAT	The operation did not finish because a **quick format** is not possible.
0xC10003FC LDM_E_IO_ERROR	The operation did not finish because an I/O error occurred. Check the System Event Log for more information.
0xC10003FD LDM_E_NO_DRIVE_LETTER	The operation requires that a drive letter be assigned to the volume.
0xC10003FE LDM_E_FILE_NOT_FOUND	The operation did not finish because the required file is not present.

Copyright © 2016 Microsoft Corporation

Return value/code	Description
0xC10003FF LDM_E_CANNOT_LOAD	The restore operation did not finish because either the floppy disk does not contain the configuration information or the information is corrupted.
0xC1000400 LDM_E_MOUNTPOINT_BUSY	The drive letter could not be assigned because it is already in use.
0xC1000401 LDM_E_FORMAT_FAILED	The format did not finish successfully.
0xC1000402 LDM_E_SERVER_NOTREADY	The LDM Administrative Service is not yet ready to accept connections.
0xC1000403 LDM_E_CANT_PROTECT_SYSTEM	The LDM cannot secure the **system partition**.
0xC1000404 LDM_E_CANT_UNPROTECT_SYSTEM	The LDM cannot unsecure the system partition.
0xC1000405 LDM_E_FAILED_SHUTDOWN	The LDM cannot shut down the system.
0xC1000406 LDM_E_VOL_TOO_BIG	The volume size is too big for the selected file system.
0xC1000407 LDM_E_VOL_TOO_SMALL	The volume size is too small for the selected file system.
0xC1000408 LDM_CLUSTER_SIZE_TOO_BIG	The cluster size is too big for the selected file system.
0xC1000409 LDM_CLUSTER_SIZE_TOO_SMALL	The cluster size is too small for the selected file system.
0x8100040A LDM_W_CANTOPENLOG	Debug log file "%1" could not be opened. Debug tracing is not available.
0x8100040B LDM_W_TIMEOUT	The request timed out and could not be finished.
0x8100040C LDM_W_UNSUPPORTED	The requested operation is not supported.
0x0100040D LDM_S_REBOOT_PENDING	The drive letter reassignment will not occur until the computer is restarted.
0x8100040E LDM_W_VOL_COMPRESS_FAILED	The format succeeded, but file and folder compression are not enabled.
0xC1000411 LDM_E_VMCONFIG_LOAD_FAILED	Windows cannot load the following LDM configuration library: "%1.dll". The LDM might not be correctly installed, or the system folder might be corrupted. Contact the system administrator for assistance.
0xC1000412 LDM_E_VOLUMEDISABLED	The requested operation is not supported on failed volumes.
0xC1000413	The requested operation cannot be finished because

Copyright © 2016 Microsoft Corporation

Return value/code	Description
LDM_E_RDONLY	the media is write-protected.
0xC1000414 LDM_E_NO_DYNAMIC	Dynamic disks are not supported on this system.
0xC1000415 LDM_E_INVALID_FS	The path provided is on a file system that does not support drive paths.
0xC1000416 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.
0xC1000417 LDM_E_DIR_NOT_EMPTY	The path cannot be used for creating a drive path because the folder is not empty.
0xC1000418 LDM_E_NOT_DRIVE_PATH	The LDM cannot delete the drive path.
0xC1000419 LDM_E_INVALID_NAME	The path is not valid and cannot be used for creating a drive path.
0xC100041B LDM_E_MEDIA_DOESNT_SUPPORT_MOUNT_POINTS	Drive paths are not supported for **removable media**.
0xC100041C LDM_E_DELETE_NOACCESS	The drive path cannot be deleted because access is denied.
0xC100041D LDM_E_VOLUME_DISABLED	The operation did not finish because the partition or volume is not enabled. To enable the partition or volume, restart the computer.
0xC100041E LDM_CLUSTER_COUNT_TOO_HIGH	The format operation did not finish because the cluster count is higher than expected.
0x8100041F 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.
0x81000420 LDM_W_MARK_ACITVE_FAILED_LOGICAL	The volume was repaired but the underlying partition was not marked active because it is a **logical drive**.
0xC1000421 LDM_E_SERVICE_DISABLED	The LDM Administrative Service is disabled.
0xC1000422 LDM_E_BOOTFILE	The LDM could not update the **boot file** for any **boot partitions** on the destination disk. Verify the ARC path listings in file **boot.ini** or through the bootcfg.exe tool.
0xC1000423 LDM_E_BAD_HARDWARE	The disk configuration operation did not finish. Check the System Event Log for more information on the error. Verify the status of the storage devices before retrying. If that does not solve the problem, close the Disk Management console, then restart Disk Management or restart the computer.
0xC1000424 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

Copyright © 2016 Microsoft Corporation

Return value/code	Description
	controller. Unable to update the boot file. Verify the arcpath listings in file boot.ini or through the bootcfg.exe tool.
0xC1000425 LDM_E_VOLUME_GROW_FAILED_FS	Failed to extend the file system for the volume.
0xC1000426 LDM_E_GPT_BOOT_MIRRORED_TO_MBR	The current **boot volume** has been mirrored to a **master boot record (MBR)** disk. It will not be possible to boot from the plex on the MBR disk.
0xC1000427 LDM_E_REGION_LATENCY_RETRY	The cache has not completely updated after adding or removing partitions. Retry the operation.
0xC10007CA LDM_E_GENERIC_ERROR	The operation did not finish. Check the System Event Log for more information on the error.
0xC10007CB LDM_E_GENERIC_RETRY	The operation did not finish. Check the System Event Log for more information on the error. Retrying the operation may fix the problem.
0xC10007CC LDM_E_GENERIC_RESTART	The operation did not finish. Check the System Event Log for more information on the error. Close the Disk Management console, then restart Disk Management before retrying the operation.
0xC10007CD LDM_E_GENERIC_REBOOT	The operation did not finish. Check the System Event Log for more information on the error. Restart the computer before retrying the operation.
0xC10007CE LDM_E_NO_VM	No dynamic disks are present.
0xC10007CF LDM_E_INTERNALFAILURE	An internal error has occurred. Close the Disk Management console, then restart Disk Management, or restart the computer.
0xC10007D0 LDM_E_PAGEFILE_VOLUME	This volume contains a pagefile.
0xC10007D1 LDM_E_SYSTEM_VOLUME	This volume is marked as an active (system) volume.
0xC10007D2 LDM_E_CRASHDUMP_VOLUME	This volume is configured to hold a "crashdump" file.
0xC10007D3 LDM_E_CRASHDUMP_PAGEFILE_BOOT_SYSTEM_VOLUME	The request cannot be finished because the volume is open or in use. It ~~may~~can be configured as a system, boot, or pagefile volume, or configured 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.

Copyright © 2016 Microsoft Corporation

2.2.3 REGIONTYPE

The REGIONTYPE enumeration defines values for region types.

```
typedef  enum _REGIONTYPE
{
  REGION_UNKNOWN,
  REGION_FREE,
  REGION_EXTENDED_FREE,
  REGION_PRIMARY,
  REGION_LOGICAL,
  REGION_EXTENDED,
  REGION_SUBDISK,
  REGION_CDROM,
  REGION_REMOVABLE
} 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,
  VOLUMETYPE_PRIMARY_PARTITION,
  VOLUMETYPE_LOGICAL_DRIVE,
  VOLUMETYPE_FT,
  VOLUMETYPE_VM,
  VOLUMETYPE_CDROM,
  VOLUMETYPE_REMOVABLE
} 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 operating system–style **fault-tolerant** volume.

Copyright © 2016 Microsoft Corporation

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,
  VOLUMELAYOUT_PARTITION,
  VOLUMELAYOUT_SIMPLE,
  VOLUMELAYOUT_SPANNED,
  VOLUMELAYOUT_MIRROR,
  VOLUMELAYOUT_STRIPE,
  VOLUMELAYOUT_RAID5
} 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,
  REQ_STARTED,
  REQ_IN_PROGRESS,
  REQ_COMPLETED,
  REQ_ABORTED,
  REQ_FAILED
} REQSTATUS;
```

REQ_UNKNOWN: Request state is unknown.

REQ_STARTED: Request has started.

REQ_IN_PROGRESS: Request is in progress.

REQ_COMPLETED: Request has finished.

REQ_ABORTED: Request has terminated.

REQ_FAILED: Request has failed.

Copyright © 2016 Microsoft Corporation

2.2.7 REGIONSTATUS

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

```
typedef  enum _REGIONSTATUS
{
  REGIONSTATUS_UNKNOWN,
  REGIONSTATUS_OK,
  REGIONSTATUS_FAILED,
  REGIONSTATUS_FAILING,
  REGIONSTATUS_REGENERATING,
  REGIONSTATUS_NEEDSRESYNC
} 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,
  VOLUME_STATUS_HEALTHY,
  VOLUME_STATUS_FAILED,
  VOLUME_STATUS_FAILED_REDUNDANCY,
  VOLUME_STATUS_FAILING,
  VOLUME_STATUS_FAILING_REDUNDANCY,
  VOLUME_STATUS_FAILED_REDUNDANCY_FAILING,
  VOLUME_STATUS_SYNCHING,
  VOLUME_STATUS_REGENERATING,
  VOLUME_STATUS_INITIALIZING,
  VOLUME_STATUS_FORMATTING
} 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.

Copyright © 2016 Microsoft Corporation

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 (UID) 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 VOLUME_SPEC structure specifies a new volume to create. VOLUME_SPEC is a typedef of this structure.

```
typedef struct volumespec {
  VOLUMETYPE type;
  VOLUMELAYOUT layout;
  REGIONTYPE partitionType;
  LONGLONG length;
  LONGLONG lastKnownState;
}
typedef struct volumespec 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;
  VOLUMELAYOUT layout;
  LONGLONG length;
  LdmObjectId fsId;
```


Copyright © 2016 Microsoft Corporation

```
    unsigned long memberCount;
    VOLUMESTATUS status;
    LONGLONG lastKnownState;
    LdmObjectId taskId;
    unsigned long vflags;
  }
  typedef struct volumeinfo VOLUME_INFO;
```

id: Specifies the **object identifier (OID)** 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 following applicable flags 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	Volume contains a **crash dump file**.

Copyright © 2016 Microsoft Corporation

Value	Meaning
0x00008000	
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.

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

diskId: Specifies the OID 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.

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

regionId: Specifies the OID for the region.

Copyright © 2016 Microsoft Corporation

regionType: Specifies the region type.

diskId: Specifies the OID 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.

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

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

storageId: Specifies the OID 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 as follows with a logical OR operation.

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

Copyright © 2016 Microsoft Corporation

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].

```
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;
}
};
typedef struct filesysteminfo FILE_SYSTEM_INFO;
```

id: Specifies the OID for the file system.

storageId: Specifies the OID 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 **NT 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	File system type is a **FAT32 file system**.

Copyright © 2016 Microsoft Corporation

Value	Meaning
0x00000003	
FSTYPE_CDFS 0x00000004	File system type is **Compact Disc 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 the terminating null character.

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].

```
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;
+
};
typedef struct ifilesysteminfo 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 file system.
FSTYPE_CDFS 0x00000004	File system type is CDFS.
FSTYPE_UDF 0x00000005	File system type is **UDF**.
FSTYPE_OTHER	File system type is not listed.

Copyright © 2016 Microsoft Corporation

Value	Meaning
0x80000000	

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.
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.
FSF_FS_ALLOC_SZ_2K 0x00000100	File system supports an allocation unit size of 2 kilobytes.
FSF_FS_ALLOC_SZ_4K 0x00000200	File system supports an allocation unit size of 4 kilobytes.
FSF_FS_ALLOC_SZ_8K 0x00000400	File system supports an allocation unit size of 8 kilobytes.
FSF_FS_ALLOC_SZ_16K 0x00000800	File system supports an allocation unit size of 16 kilobytes.
FSF_FS_ALLOC_SZ_32K 0x00001000	File system supports an allocation unit size of 32 kilobytes.
FSF_FS_ALLOC_SZ_64K 0x00002000	File system supports an allocation unit size of 64 kilobytes.
FSF_FS_ALLOC_SZ_128K 0x00004000	File system supports an allocation unit size of 128 kilobytes.
FSF_FS_ALLOC_SZ_256K 0x00008000	File system supports an allocation unit size of 256 kilobytes.

Copyright © 2016 Microsoft Corporation

Value	Meaning
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 kilobyte.
FSF_FS_ALLOC_SZ_2K 0x00000100	File system supports an allocation unit size of 2 kilobytes.
FSF_FS_ALLOC_SZ_4K 0x00000200	File system supports an allocation unit size of 4 kilobytes.
FSF_FS_ALLOC_SZ_8K 0x00000400	File system supports an allocation unit size of 8 kilobytes.
FSF_FS_ALLOC_SZ_16K 0x00000800	File system supports an allocation unit size of 16 kilobytes.
FSF_FS_ALLOC_SZ_32K 0x00001000	File system supports an allocation unit size of 32 kilobytes.
FSF_FS_ALLOC_SZ_64K 0x00002000	File system supports an allocation unit size of 64 kilobytes.
FSF_FS_ALLOC_SZ_128K 0x00004000	File system supports an allocation unit size of 128 kilobytes.
FSF_FS_ALLOC_SZ_256K 0x00008000	File system supports an allocation unit size of 256 kilobytes.
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.

```
struct taskinfo {
```


Copyright © 2016 Microsoft Corporation

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

id: Specifies the OID for the task.

storageId: Specifies the OID 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 OID of the client that requested the task.

percentComplete: Percentage of the task that is complete. 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,
  PROGRESS_FORMAT,
  PROGRESS_SYNCHING
} 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**.

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

sourceId: Specifies the OID of the source volume. The source volume has a folder to which the target volume will be mounted.

targetId: Specifies the OID of the target volume.

cchString: Specifies the length of the **mount path**, including the terminating null character.

sstring: Null-terminated Unicode string that contains the mount path of the source.

2.2.20 MERGE_OBJECT_INFO

The MERGE_OBJECT_INFO structure provides change information for a merge operation.

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

Copyright © 2016 Microsoft Corporation

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.

```
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;
  [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;
}
};
typedef struct diskinfo DISK_INFO;
```

id: Specifies the OID 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.

Copyright © 2016 Microsoft Corporation

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 0x00000010	Disk is not ready yet.
DEVICESTATE_MISSING	Disk is no longer available.

Copyright © 2016 Microsoft Corporation

Value	Meaning
0x00000020	
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.
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 **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 whether the disk can be converted to a dynamic disk. Will be true if the disk is basic, healthy, and has 512 byte sectors.

Copyright © 2016 Microsoft Corporation

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 the terminating null character.

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

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

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

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

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; for 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.

```
struct regioninfo {
  LdmObjectId id;
  LdmObjectId diskId;
  LdmObjectId volId;
  LdmObjectId fsId;
  LONGLONG start;
  LONGLONG length;
  REGIONTYPE regionType;
```


Copyright © 2016 Microsoft Corporation

```
    unsigned long partitionType;
    boolean isActive;
    REGIONSTATUS status;
    hyper lastKnownState;
    LdmObjectId taskId;
    unsigned long rflags;
    unsigned long currentPartitionNumber;
  }
  };
  typedef struct regioninfo REGION_INFO;
```

id: Specifies the region's OID.

diskId: Specifies the OID of the disk on which the region resides.

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

fsId: Specifies the OID 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. This field contains one of the following values.

Value	Meaning
PARTITION_ENTRY_UNUSED 0x00	An unused entry partition.
PARTITION_EXTENDED 0x05	An extended partition.
PARTITION_FAT_12 0x01	A **FAT12 file system** partition.
PARTITION_FAT_16 0x04	A FAT16 file system partition.
PARTITION_FAT32 0x0B	A FAT32 file system partition.
PARTITION_IFS 0x07	An installable file system (IFS) partition.
PARTITION_LDM 0x42	An LDM partition.
PARTITION_NTFT 0x80	A Windows NT operating system fault-tolerant (FT) partition.
VALID_NTFT 0xC0	A valid Windows NT FT partition. The high bit of a partition type code indicates that a partition is part of an NT FT mirror or striped array.

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

Copyright © 2016 Microsoft Corporation

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 0x00000001	Region is currently being formatted.
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.

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;
```


Copyright © 2016 Microsoft Corporation

PARTITIONSTYLE_UNKNOWN: Partition is of an unknown style.

PARTITIONSTYLE_MBR: Partition is of the MBR style.

PARTITIONSTYLE_GPT: Partition is of the GPT style.

2.5.1.2 DISK_INFO_EX

The DISK_INFO_EX structure provides information about a disk.

```
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;
```

id: Specifies the OID of the disk.

length: Size of the disk in bytes.

freeBytes: Number of unallocated bytes on the disk.

Copyright © 2016 Microsoft Corporation

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 0x000000010	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	Disk is fully functional.

Copyright © 2016 Microsoft Corporation

Value	Meaning
0x00000001	
DEVICESTATE_NO_MEDIA 0x00000002	Disk has no media.
DEVICESTATE_NOSIG 0x00000004	Disk has an invalid signature.
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 IDE bus.
BUSTYPE_SCSI 0x00000002	Disk resides on an SCSI bus.
BUSTYPE_FIBRE 0x00000003	Disk resides on a fiber channel bus.
BUSTYPE_USB 0x00000004	Disk resides on a USB.
BUSTYPE_SSA 0x00000005	Disk resides on an SSA bus.
BUSTYPE_1394 0x00000006	Disk resides on an IEEE 1394 bus.

attributes: Bitmap of disk attributes.

Copyright © 2016 Microsoft Corporation

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

partitionStyle: Value from the PARTITIONSTYLE enumeration that indicates the disk's partitioning style.

(unnamed union): 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, as specified in [MS-DTYP], section 2.3.4.1, of the disk.<9>

portNumber: SCSI port number of the disk.

targetNumber: SCSI target identifier of the disk.

lunNumber: SCSI 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 the terminating null character.

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

Copyright © 2016 Microsoft Corporation

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

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

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

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

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**, as specified in [MS-DTYP] section 2.3.4.3, appended to the end. For example: "\\?\ide#diskwdc_wd1600jd-75hbb0______________________08.02d08#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.

```
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;
```


Copyright © 2016 Microsoft Corporation

id: Specifies the region's OID.

diskId: Specifies the OID of the disk on which the region resides.

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

fsId: Specifies the OID 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.

(unnamed union): 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 3.1 operating system, Windows NT 3.5 operating system, Windows NT 3.51 operating system, and Windows NT 4.0 partition style for the region. This field contains one of the following values.

Value	Meaning
PARTITION_ENTRY_UNUSED 0x00	An unused entry partition.
PARTITION_EXTENDED 0x05	An extended partition.
PARTITION_FAT_12 0x01	A FAT12 file system partition.
PARTITION_FAT_16 0x04	A FAT16 file system partition.
PARTITION_FAT32 0x0B	A FAT32 file system partition.
PARTITION_IFS 0x07	An IFS partition.
PARTITION_LDM 0x42	An LDM partition.
PARTITION_NTFT 0x80	A Windows NT fault-tolerant (FT) partition.
VALID_NTFT 0xC0	A valid Windows NT FT partition. The high bit of a partition type code indicates that a partition is part of an NTFT mirror or striped array.

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

Copyright © 2016 Microsoft Corporation

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

partitionType: Windows NT partition style for the disk. This field contains one of the following values.

Value	Meaning
PARTITION_BASIC_DATA_GUID ebd0a0a2-b9e5-4433-87c0-68b6b72699c7	The data partition type that is created and recognized by Windows.
PARTITION_ENTRY_UNUSED_GUID 00000000-0000-0000-0000-000000000000	There is no partition.
PARTITION_SYSTEM_GUID c12a7328-f81f-11d2-ba4b-00a0c93ec93b	The partition is an Extensible Firmware Interface (EFI) system partition.
PARTITION_MSFT_RESERVED_GUID e3c9e316-0b5c-4db8-817d-f92df00215ae	The partition is a Microsoft reserved partition.
PARTITION_LDM_METADATA_GUID 5808c8aa-7e8f-42e0-85d2-e1e90434cfb3	The partition is an LDM metadata partition on a dynamic disk.
PARTITION_LDM_DATA_GUID af9b60a0-1431-4f62-bc68-3311714a69ad	The partition is an LDM data partition on a dynamic disk.
PARTITION_MSFT_RECOVERY_GUID de94bba4-06d1-4d40-a16a-bfd50179d6ac	The partition is a Microsoft recovery partition.

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	Region contains the paging file.

Copyright © 2016 Microsoft Corporation

Value	Meaning
0x00000004	
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.

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

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
```


Copyright © 2016 Microsoft Corporation

```
{
  DMNOTIFY_UNKNOWN_INFO,
  DMNOTIFY_DISK_INFO,
  DMNOTIFY_VOLUME_INFO,
  DMNOTIFY_REGION_INFO,
  DMNOTIFY_TASK_INFO,
  DMNOTIFY_DL_INFO,
  DMNOTIFY_FS_INFO,
  DMNOTIFY_SYSTEM_INFO
} 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,
  LDMACTION_CREATED,
  LDMACTION_DELETED,
  LDMACTION_MODIFIED,
  LDMACTION_FAILED
} 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.

Copyright © 2016 Microsoft Corporation

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 fall back 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 3.2.4.

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, which are listed in opnum order.

Method	Description
IVolumeClient::EnumDisks	Opnum: 3

Copyright © 2016 Microsoft Corporation

Method	Description
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
IVolumeClient::FTEnumVolumes	Opnum: 13
IVolumeClient::FTEnumLogicalDiskMembers	Opnum: 14
IVolumeClient::FTDeleteVolume	Opnum: 15
IVolumeClient::FTBreakMirror	Opnum: 16
IVolumeClient::FTResyncMirror	Opnum: 17
IVolumeClient::FTRegenerateParityStripe	Opnum: 18
IVolumeClient::FTReplaceMirrorPartition	Opnum: 19
IVolumeClient::FTReplaceParityStripePartition	Opnum: 20
IVolumeClient::EnumDriveLetters	Opnum: 21
IVolumeClient::AssignDriveLetter	Opnum: 22
IVolumeClient::FreeDriveLetter	Opnum: 23
IVolumeClient::EnumLocalFileSystems	Opnum: 24
IVolumeClient::GetInstalledFileSystems	Opnum: 25
IVolumeClient::Format	Opnum: 26
IVolumeClient::EnumVolumes	Opnum: 28
IVolumeClient::EnumVolumeMembers	Opnum: 29
IVolumeClient::CreateVolume	Opnum: 30
IVolumeClient::CreateVolumeAssignAndFormat	Opnum: 31
IVolumeClient::CreateVolumeAssignAndFormatEx	Opnum: 32
IVolumeClient::GetVolumeMountName	Opnum: 33
IVolumeClient::GrowVolume	Opnum: 34
IVolumeClient::DeleteVolume	Opnum: 35
IVolumeClient::AddMirror	Opnum: 36
IVolumeClient::RemoveMirror	Opnum: 37

Copyright © 2016 Microsoft Corporation

Method	Description
IVolumeClient::SplitMirror	Opnum: 38
IVolumeClient::InitializeDisk	Opnum: 39
IVolumeClient::UninitializeDisk	Opnum: 40
IVolumeClient::ReConnectDisk	Opnum: 41
IVolumeClient::ImportDiskGroup	Opnum: 43
IVolumeClient::DiskMergeQuery	Opnum: 44
IVolumeClient::DiskMerge	Opnum: 45
IVolumeClient::ReAttachDisk	Opnum: 47
IVolumeClient::ReplaceRaid5Column	Opnum: 51
IVolumeClient::RestartVolume	Opnum: 52
IVolumeClient::GetEncapsulateDiskInfo	Opnum: 53
IVolumeClient::EncapsulateDisk	Opnum: 54
IVolumeClient::QueryChangePartitionNumbers	Opnum: 55
IVolumeClient::DeletePartitionNumberInfoFromRegistry	Opnum: 56
IVolumeClient::SetDontShow	Opnum: 57
IVolumeClient::GetDontShow	Opnum: 58
IVolumeClient::EnumTasks	Opnum: 67
IVolumeClient::GetTaskDetail	Opnum: 68
IVolumeClient::AbortTask	Opnum: 69
IVolumeClient::HrGetErrorData	Opnum: 70
IVolumeClient::Initialize	Opnum: 71
IVolumeClient::Uninitialize	Opnum: 72
IVolumeClient::Refresh	Opnum: 73
IVolumeClient::RescanDisks	Opnum: 74
IVolumeClient::RefreshFileSys	Opnum: 75
IVolumeClient::SecureSystemPartition	Opnum: 76
IVolumeClient::ShutDownSystem	Opnum: 77
IVolumeClient::EnumAccessPath	Opnum: 78
IVolumeClient::EnumAccessPathForVolume	Opnum: 79
IVolumeClient::AddAccessPath	Opnum: 80
IVolumeClient::DeleteAccessPath	Opnum: 81
IVolumeClient2::GetMaxAdjustedFreeSpace	Opnum: 3

Copyright © 2016 Microsoft Corporation

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
IVolumeClient3::FTEnumVolumes	Opnum: 13
IVolumeClient3::FTEnumLogicalDiskMembers	Opnum: 14
IVolumeClient3::FTDeleteVolume	Opnum: 15
IVolumeClient3::FTBreakMirror	Opnum: 16
IVolumeClient3::FTResyncMirror	Opnum: 17
IVolumeClient3::FTRegenerateParityStripe	Opnum: 18
IVolumeClient3::FTReplaceMirrorPartition	Opnum: 19
IVolumeClient3::FTReplaceParityStripePartition	Opnum: 20
IVolumeClient3::EnumDriveLetters	Opnum: 21
IVolumeClient3::AssignDriveLetter	Opnum: 22
IVolumeClient3::FreeDriveLetter	Opnum: 23
IVolumeClient3::EnumLocalFileSystems	Opnum: 24
IVolumeClient3::GetInstalledFileSystems	Opnum: 25
IVolumeClient3::Format	Opnum: 26
IVolumeClient3::EnumVolumes	Opnum: 27
IVolumeClient3::EnumVolumeMembers	Opnum:28
IVolumeClient3::CreateVolume	Opnum: 29
IVolumeClient3::CreateVolumeAssignAndFormat	Opnum: 30
IVolumeClient3::CreateVolumeAssignAndFormatEx	Opnum: 31
IVolumeClient3::GetVolumeMountName	Opnum: 32
IVolumeClient3::GrowVolume	Opnum: 33
IVolumeClient3::DeleteVolume	Opnum: 34
IVolumeClient3::CreatePartitionsForVolume	Opnum: 35

Copyright © 2016 Microsoft Corporation

Method	Description
IVolumeClient3::DeletePartitionsForVolume	Opnum: 36
IVolumeClient3::GetMaxAdjustedFreeSpace	Opnum: 37
IVolumeClient3::AddMirror	Opnum: 38
IVolumeClient3::RemoveMirror	Opnum: 39
IVolumeClient3::SplitMirror	Opnum: 40
IVolumeClient3::InitializeDiskEx	Opnum: 41
IVolumeClient3::UninitializeDisk	Opnum: 42
IVolumeClient3::ReConnectDisk	Opnum: 43
IVolumeClient3::ImportDiskGroup	Opnum: 44
IVolumeClient3::DiskMergeQuery	Opnum: 45
IVolumeClient3::DiskMerge	Opnum: 46
IVolumeClient3::ReAttachDisk	Opnum: 47
IVolumeClient3::ReplaceRaid5Column	Opnum: 48
IVolumeClient3::RestartVolume	Opnum: 49
IVolumeClient3::GetEncapsulateDiskInfoEx	Opnum: 50
IVolumeClient3::EncapsulateDiskEx	Opnum: 51
IVolumeClient3::QueryChangePartitionNumbers	Opnum: 52
IVolumeClient3::DeletePartitionNumberInfoFromRegistry	Opnum: 53
IVolumeClient3::SetDontShow	Opnum: 54
IVolumeClient3::GetDontShow	Opnum: 55
IVolumeClient3::EnumTasks	Opnum: 64
IVolumeClient3::GetTaskDetail	Opnum: 65
IVolumeClient3::AbortTask	Opnum: 66
IVolumeClient3::HrGetErrorData	Opnum: 67
IVolumeClient3::Initialize	Opnum: 68
IVolumeClient::Uninitialize	Opnum: 69
IVolumeClient3::Refresh	Opnum: 70
IVolumeClient3::RescanDisks	Opnum: 71
IVolumeClient3::RefreshFileSys	Opnum: 72
IVolumeClient3::SecureSystemPartition	Opnum: 73
IVolumeClient3::ShutDownSystem	Opnum: 74
IVolumeClient3::EnumAccessPath	Opnum: 75

Copyright © 2016 Microsoft Corporation

Method	Description
IVolumeClient3::EnumAccessPathForVolume	Opnum: 76
IVolumeClient3::AddAccessPath	Opnum: 77
IVolumeClient3::DeleteAccessPath	Opnum: 78
IVolumeClient4::RefreshEx	Opnum: 3
IVolumeClient4::GetVolumeDeviceName	Opnum: 4
IDMRemoteServer::CreateRemoteObject	Opnum: 3
IDMNotify::ObjectsChanged	Opnum: 3

The server MUST implement all of the preceding methods of the IVolumeClient and IVolumeClient2 interfaces.

The server SHOULD implement all of the preceding methods of the 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 the preceding 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 specification. 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 in the following sections, when calling methods in IVolumeClient3 rather than IVolumeClient, references to the DISK_INFO structure MUST be replaced with

Copyright © 2016 Microsoft Corporation

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 structure 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: Before 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. The EnumDiskRegions method returns these values as the **REGION_INFO::volId** and **REGION_INFO::lastKnownState** structure members. For volumes on basic disks, the region's **lastKnownState** is the same as the volume's

Copyright © 2016 Microsoft Corporation

lastKnownState. FTEnumVolumes and EnumVolumes return these parameters as the **VOLUME_INFO::id** and **VOLUME_INFO::lastKnownState** structure members.

Before invoking AssignDriveLetter, the client MUST also invoke EnumDriveLetters. The EnumDriveLetters method returns 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. The EnumDriveLetters method 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 whether the drive letter is already in use by some other volume.

FreeDriveLetter: The client MUST use the preceding call sequence description for AssignDriveLetter, except that in the final step the client MUST use FreeDriveLetter, rather than AssignDriveLetter, to free the drive letter.

3.1.4.1.3 File Systems

AddAccessPath: Before 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. The EnumDiskRegions method returns this value as the **REGION_INFO::volId** structure member. The FTEnumVolumes and EnumVolumes methods return this parameter as the **VOLUME_INFO::id** structure members.

DeleteAccessPath: The client MUST use the preceding call sequence description above for AddAccessPath, except that in the final step the client MUST use DeleteAccessPath rather than AddAccessPath.

Format: Before 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. The EnumDiskRegions method returns these values as the **REGION_INFO::volId** and **REGION_INFO::lastKnownState** structure members. For volumes on basic disks, the region's **lastKnownState** is the same as the volume's **lastKnownState**. The FTEnumVolumes and EnumVolumes methods return these parameters as the **VOLUME_INFO::id** and **VOLUME_INFO::lastKnownState** structure members.

Before 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. The GetInstalledFileSystems method 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: Before 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. The EnumDiskRegions method returns this value as the **REGION_INFO::volId** structure member. The FTEnumVolumes and EnumVolumes methods return this parameter as the **VOLUME_INFO::id** structure members.

EnumAccessPathForVolume: The client MUST use the preceding call sequence description for GetVolumeMountName, except that in the final step it MUST use EnumAccessPathForVolume rather than GetVolumeMountName.

3.1.4.1.4 Disks

Copyright © 2016 Microsoft Corporation

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.

Before 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. The DiskMergeQuery method returns these values as the *merge_config_tid*, *merge_dm_rids*, and *numRids* output parameters.

DiskMergeQuery: Before 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. The EnumDisks method 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. The EnumDisks method 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: Before 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. The EnumDisks method returns these values as the **DISK_INFO::id** and **DISK_INFO::lastKnownState** structure members.

EncapsulateDisk: Before 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. The GetEncapsulateDiskInfo method 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. If the *affectedVolumeCount* returned by GetEncapsulateDiskInfo is zero, the client MUST allocate at least 1 byte for *affectedVolumeList* before passing it to EncapsulateDisk method. If the *affectedRegionCount* returned by GetEncapsulateDiskInfo is zero, the client MUST allocate at least 1 byte for *affectedRegionList* before passing it to the EncapsulateDisk method.

EncapsulateDiskEx: Before 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. The GetEncapsulateDiskInfoEx method 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. If the *affectedVolumeCount* returned by GetEncapsulateDiskInfoEx is zero, the client MUST allocate at least 1 byte for *affectedVolumeList* before passing it to EncapsulateDiskEx method. If the *affectedRegionCount* returned by GetEncapsulateDiskInfoEx is zero, the client MUST allocate at least 1 byte for *affectedRegionList* before passing it to the EncapsulateDiskEx method.

EnumDiskRegions: Before 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. The EnumDisks method returns this value as the **DISK_INFO::id** structure member.

Copyright © 2016 Microsoft Corporation

EnumDiskRegionsEx: Before 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. The EnumDisksEx method returns this value as the **DISK_INFO_EX::id** structure member.

GetEncapsulateDiskInfo: Before 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: Before 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 preceding call sequence description for EnumDiskRegions, except that in the final step it MUST use GetMaxAdjustedFreeSpace rather than EnumDiskRegions.

ImportDiskGroup: Before 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. The EnumDisks method returns these values as the **DISK_INFO::dgid** and **DISK_INFO::cchDgid** structure members.

InitializeDisk: Before 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. The EnumDisks method returns these values as the **DISK_INFO::id** and **DISK_INFO::lastKnownState** structure members.

InitializeDiskEx: The client MUST use the following call sequence description for InitializeDiskStyle, except that in the final step the client MUST use InitializeDiskEx rather than 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. The EnumDisksEx method 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 whether 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 rather than WriteSignature.

ReConnectDisk: Before 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.

UninitializeDisk: Before 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. The EnumDisks method returns these values as the **DISK_INFO::id** and **DISK_INFO::lastKnownState** structure members.

Copyright © 2016 Microsoft Corporation

WriteSignature: Before 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. The EnumDisks method returns these values as the **DISK_INFO::id** and **DISK_INFO::lastKnownState** structure members.

3.1.4.1.5 Partitions

CreatePartition: Before invoking CreatePartition, the client MUST invoke EnumDiskRegions to retrieve the region ID, the disk ID, and the region's last known state. The client MUST pass these returned values as the REGION_SPEC structure's **regionId**, **diskId**, and **lastKnownState** members in the input parameter to the CreatePartition method. These values are returned by EnumDiskRegions as *REGION_INFO::id*, *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 preceding call sequence description for CreatePartition, except that in the final step the client MUST use DeletePartition rather than CreatePartition.

EnumVolumeMembers: The client MUST use the preceding call sequence description for GetVolumeMountName, except that in the final step the client MUST use EnumVolumeMembers rather than GetVolumeMountName.

MarkActivePartition: Before 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. The EnumDiskRegions method 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 the 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 the client 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 whether 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.

Copyright © 2016 Microsoft Corporation

CreatePartitionsForVolume: The client MUST use the preceding call sequence description for DeleteVolume, except that in the final step it MUST use CreatePartitionsForVolume rather than DeleteVolume.

CreateVolume: Before 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 determine whether 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 the client MUST NOT call CreateVolume. If the client does not want to assign a drive letter, the letter parameter MUST be a 2-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 2-byte null character or Unicode SPACE character. If the client 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 the client 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 the client 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 the client MUST use DeletePartitionsForVolume rather than DeleteVolume.

DeleteVolume: Before invoking DeleteVolume, the client MUST invoke 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. The EnumVolumes method 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 the client MUST use FTBreakMirror rather than FTDeleteVolume.

FTDeleteVolume: Before invoking FTDeleteVolume, the client MUST invoke FTEnumVolumes 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. The FTEnumVolumes method returns these parameters as the **VOLUME_INFO::id** and **VOLUME_INFO::lastKnownState** structure members.

FTEnumLogicalDiskMembers: Before 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. The

Copyright © 2016 Microsoft Corporation

EnumDiskRegions method returns this value as the **REGION_INFO::volId** structure member; FTEnumVolumes returns this parameter as the **VOLUME_INFO::id** structure member.

FTRegenerateParityStripe: The client MUST use the preceding call sequence description for FTDeleteVolume, except that in the final step the client MUST use FTRegenerateParityStripe rather than FTDeleteVolume.

FTReplaceMirrorPartition: Before invoking FTReplaceMirrorPartition, the client MUST invoke FTEnumVolumes 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. FTEnumVolumes returns these parameters as the **VOLUME_INFO::id** and **VOLUME_INFO::lastKnownState** structure members.

Before 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. The EnumDiskRegions method 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 because they are not used or implemented.

FTReplaceParityStripePartition: The client MUST use the preceding call sequence description for FTReplaceMirrorPartition, except that in the final step the client MUST use FTReplaceParityStripePartition rather than FTReplaceMirrorPartition.

FTResyncMirror: The client MUST use the preceding call sequence description for FTDeleteVolume, except that in the final step the client MUST use FTResyncMirror rather than FTDeleteVolume.

GetVolumeDeviceName: The client MUST use the preceding call sequence description for GetVolumeMountName, except that in the final step it MUST use GetVolumeDeviceName rather than 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.

Before 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 the client 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 the client 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.

Before 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.

Copyright © 2016 Microsoft Corporation

The EnumDisks method 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 rather than 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 the volume's last known state input parameters, except that in the final step the client 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 the client MUST NOT call WriteSignature. If the client does not want to assign a drive letter, the letter parameter MUST be a 2-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 the client 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: Before 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. The EnumTasks method 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 rather than 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 previous 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 ([MS-ERREF] section 2.1). 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>

Copyright © 2016 Microsoft Corporation

Notifications related to storage object modification indicate a state change, such as the change of a region's status from REGIONSTATUS_OK to REGIONSTATUS_FAILED, or a change in a volume's length 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 finished without errors. If the value of the status field is REQ_ABORTED, this indicates that the server processing was interrupted and did not finish successfully. If the value of the status field is REQ_FAILED, this indicates that the server processing failed and did not finish 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 finished its processing for the operation. The client MUST NOT assume the server's processing is finished 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 states 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)

Copyright © 2016 Microsoft Corporation

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 notification 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.

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:

Copyright © 2016 Microsoft Corporation

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] section 2.1).

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**

Copyright © 2016 Microsoft Corporation

- 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 (UID) 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 need not be persisted across server sessions.<15>

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 (UID) 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.

Copyright © 2016 Microsoft Corporation

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 (UID) 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** field 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.<16>

Copyright © 2016 Microsoft Corporation

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 made 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 sections, 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 or create volume).

Copyright © 2016 Microsoft Corporation

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 SHOULD return a response to the client before the task completes.

This is the list of methods that MAY be implemented asynchronously:<17>

- *IVolumeClient::CreatePartitionAssignAndFormat*
- *IVolumeClient3::CreatePartitionAssignAndFormat*
- *IVolumeClient::CreatePartitionAssignAndFormatEx*
- *IVolumeClient3::CreatePartitionAssignAndFormatEx*
- *IVolumeClient::Format*
- *IVolumeClient3::Format*
- *IVolumeClient::CreateVolumeAssignAndFormat*
- *IVolumeClient3::CreateVolumeAssignAndFormat*
- *IVolumeClient::CreateVolumeAssignAndFormatEx*
- *IVolumeClient3::CreateVolumeAssignAndFormatEx*
- *IVolumeClient::ImportDiskGroup*
- *IVolumeClient3::ImportDiskGroup*
- *IVolumeClient::UninitializeDisk*
- *IVolumeClient3::UninitializeDisk*
- *IVolumeClient::ReConnectDisk*
- *IVolumeClient3::ReConnectDisk*

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.<18>

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:
    - 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.

Copyright © 2016 Microsoft Corporation

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::CreatePartition, IVolumeClient3::CreatePartition, IVolumeClient::DeletePartition, IVolumeClient3::DeletePartition, IVolumeClient::WriteSignature, IVolumeClient3::InitializeDiskStyle, IVolumeClient::MarkActivePartition, IVolumeClient3::MarkActivePartition, IVolumeClient::Eject IVolumeClient3::Eject, IVolumeClient::FTDeleteVolume, IVolumeClient3::FTDeleteVolume, IVolumeClient::FTBreakMirror, IVolumeClient3::FTBreakMirror, IVolumeClient::FTResyncMirror, IVolumeClient3::FTResyncMirror, IVolumeClient::FTRegenerateParityStripe, IVolumeClient3::FTRegenerateParityStripe, IVolumeClient::FTReplaceMirrorPartition, IVolumeClient3::FTReplaceMirrorPartition, IVolumeClient::FTReplaceParityStripePartition, IVolumeClient3::FTReplaceParityStripePartition, IVolumeClient::AssignDriveLetter, IVolumeClient3::AssignDriveLetter, IVolumeClient::FreeDriveLetter, IVolumeClient3::FreeDriveLetter, IVolumeClient::Format, IVolumeClient3::Format, IVolumeClient::GetEncapsulateDiskInfo, IVolumeClient3::GetEncapsulateDiskInfoEx, IVolumeClient::EnumTasks, IVolumeClient3::EnumTasks, IVolumeClient3::GetTaskDetail, and IVolumeClient::GetTaskDetail):

- Send a task completion notification to the client using the IDMNotify::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) 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:
   - Generate a unique identifier (UID) 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 IDMNotify::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 finishes with either success or failure, the server MUST send a final notification to the clients by using the IDMNotify::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.
6. When the task is finished, 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

Copyright © 2016 Microsoft Corporation

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 in the following table, all methods MUST return 0 or a nonerror HRESULT (as specified in [MS-ERREF] section 2.1) 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
IVolumeClient::FTReplaceMirrorPartition	Opnum: 19
IVolumeClient::FTReplaceParityStripePartition	Opnum: 20
IVolumeClient::EnumDriveLetters	Opnum: 21
IVolumeClient::AssignDriveLetter	Opnum: 22
IVolumeClient::FreeDriveLetter	Opnum: 23
IVolumeClient::EnumLocalFileSystems	Opnum: 24
IVolumeClient::GetInstalledFileSystems	Opnum: 25
IVolumeClient::Format	Opnum: 26

Copyright © 2016 Microsoft Corporation

Method	Description
Reserved27	Opnum: 27
IVolumeClient::EnumVolumes	Opnum: 28
IVolumeClient::EnumVolumeMembers	Opnum: 29
IVolumeClient::CreateVolume	Opnum: 30
IVolumeClient::CreateVolumeAssignAndFormat	Opnum: 31
IVolumeClient::CreateVolumeAssignAndFormatEx	Opnum: 32
IVolumeClient::GetVolumeMountName	Opnum: 33
IVolumeClient::GrowVolume	Opnum: 34
IVolumeClient::DeleteVolume	Opnum: 35
IVolumeClient::AddMirror	Opnum: 36
IVolumeClient::RemoveMirror	Opnum: 37
IVolumeClient::SplitMirror	Opnum: 38
IVolumeClient::InitializeDisk	Opnum: 39
IVolumeClient::UninitializeDisk	Opnum: 40
IVolumeClient::ReConnectDisk	Opnum: 41
Reserved_Opnum42	Opnum: 42
IVolumeClient::ImportDiskGroup	Opnum: 43
IVolumeClient::DiskMergeQuery	Opnum: 44
IVolumeClient::DiskMerge	Opnum: 45
Reserved_Opnum46	Opnum: 46
IVolumeClient::ReAttachDisk	Opnum: 47
Reserved_Opnum48	Opnum: 48
Reserved_Opnum49	Opnum: 49
Reserved_Opnum50	Opnum: 50
IVolumeClient::ReplaceRaid5Column	Opnum: 51
IVolumeClient::RestartVolume	Opnum: 52
IVolumeClient::GetEncapsulateDiskInfo	Opnum: 53
IVolumeClient::EncapsulateDisk	Opnum: 54
IVolumeClient::QueryChangePartitionNumbers	Opnum: 55
IVolumeClient::DeletePartitionNumberInfoFromRegistry	Opnum: 56
IVolumeClient::SetDontShow	Opnum: 57
IVolumeClient::GetDontShow	Opnum: 58

Copyright © 2016 Microsoft Corporation

Method	Description
Reserved0	Opnum: 59
Reserved1	Opnum: 60
Reserved2	Opnum: 61
Reserved3	Opnum: 62
Reserved4	Opnum: 63
Reserved5	Opnum: 64
Reserved6	Opnum: 65
Reserved7	Opnum: 66
IVolumeClient::EnumTasks	Opnum: 67
IVolumeClient::GetTaskDetail	Opnum: 68
IVolumeClient::AbortTask	Opnum: 69
IVolumeClient::HrGetErrorData	Opnum: 70
IVolumeClient::Initialize	Opnum: 71
IVolumeClient::Uninitialize	Opnum: 72
IVolumeClient::Refresh	Opnum: 73
IVolumeClient::RescanDisks	Opnum: 74
IVolumeClient::RefreshFileSys	Opnum: 75
IVolumeClient::SecureSystemPartition	Opnum: 76
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.

Copyright © 2016 Microsoft Corporation

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] section 2.1; 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 previously 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 OID 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] section 2.1; 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:

Copyright © 2016 Microsoft Corporation

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 previously 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] section 2.1; 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 whether partitionSpec.LastKnownState matches the **LastKnownState** field of the object.
3. Verify that the *partitionSpec.regionId* specified matches with the **regionId** field of one of the regions in the disk specified by *partitionSpec.diskId*.
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. The starting offset of the partition is specified by partitionSpec.start and

Copyright © 2016 Microsoft Corporation

the length of the partition is specified by partitionSpec.length. The type of the partition to be created is specified by the *partitionType.regionType* parameter.<19>

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 that corresponds 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 the partition is created successfully.
TASK_INFO::createTime	Not required.<20>
TASK_INFO::clientID	Not required.<21>
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.<22>

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.<23>

4. Create a new file system object that corresponds to the new partition.<24>

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.

Copyright © 2016 Microsoft Corporation

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 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] section 2.1; 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 *partitionSpec.regionId* specified matches the **regionId** field of one of the regions in the disk specified by *partitionSpec.diskId*.

4. Verify that the drive letter object specified by letter is in the list of storage objects, and check whether letterLastKnowState matches the **LastKnownState** field of the object.<25>

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. Create a partition on the free disk region specified by *partitionSpec.regionId* of the disk specified by *partitionSpec.diskId*. The starting offset of the partition is specified by partitionSpec.start and the length of the partition is specified by partitionSpec.length. The type of the partition to be created is specified by the *partitionType.regionType* partition.

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.

Copyright © 2016 Microsoft Corporation

- The **tinfo.storageId** field MUST be set to the identifier of the disk region object corresponding to the new partition.<26>

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Required.
TASK_INFO::createTime	Not required.<27>
TASK_INFO::clientID	Not required.<28>
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.<29>

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.<30> This task is asynchronous and all rules for handling asynchronous tasks apply here.

If the creation of the 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 finished, 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,
```


Copyright © 2016 Microsoft Corporation

```
    [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 the terminating null character.

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 following applicable flags with a logical OR operation.

Value	Meaning
CREATE_ASSIGN_ACCESS_PATH 0x00000001	Assign the mount point *AccessPath* 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] section 2.1; 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. This step MUST be taken after the drive letter assignment succeeds and before the format operation is attempted.

3.2.4.4.1.6 IVolumeClient::DeletePartition (Opnum 8)

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

Copyright © 2016 Microsoft Corporation

```
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 whether 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] section 2.1; 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 whether *partitionSpec.LastKnownState* matches the **LastKnownState** field of the object.<31>
3. Verify that the region type specified by *partitionSpec.regionType* matches the **regionType** field of the object.
4. Verify that the start of the partition specified by *partitionSpec.start* matches the **start** field of the object.
5. Verify that the *length* of the partition specified by *partitionSpec.length* is greater than or equal to the **length** field of the object.
6. 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.

Copyright © 2016 Microsoft Corporation

TASK_INFO member	Required for this operation
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required.<32>
TASK_INFO::clientID	Not required.<33>
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.<34>

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:<35>

1. Modify the disk object where the partition resided to account for the change in region allocation.
2. Delete the disk region object that corresponds 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] section 2.1; 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.

Copyright © 2016 Microsoft Corporation

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 MBR signature, 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::storageId	Not required.
TASK_INFO::createTime	Not required.<36>
TASK_INFO::clientID	Not required.<37>
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.<38>

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 OID 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] section 2.1; 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 whether *regionLastKnownState* matches the **LastKnownState** field of the object.

2. Verify that *tinfo* is not NULL.

Copyright © 2016 Microsoft Corporation

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.<39>
TASK_INFO::clientID	Not required.<40>
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>

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 OID of the media to eject.

diskLastKnownState: The 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.

Copyright © 2016 Microsoft Corporation

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] section 2.1; 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 whether *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.<42>
TASK_INFO::clientID	Not required.<43>
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.<44>

4. Return a response to the client that contains *tinfo* and the status of the operation.<45>

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 FT volumes on basic disks (rather than **dynamic disks**). <46>

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


Copyright © 2016 Microsoft Corporation

```
);
```

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] section 2.1; 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 preceding output parameters 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 FT volume on basic disks (rather than dynamic disks).<47>

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

volumeId: Specifies the OID 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] section 2.1; 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:

Copyright © 2016 Microsoft Corporation

- 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 OIDs in the buffer MUST be returned in the output parameter *memberCount*.
6. Return a response that contains the output parameters mentioned previously 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 FT volume specified by *volumeId* on basic disks (rather than dynamic disks). This is a synchronous task.<48>

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

volumeId: Specifies the OID of the volume to delete.

force: Boolean value that indicates whether 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] section 2.1; 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:

Copyright © 2016 Microsoft Corporation

- Verify that the FT volume specified by *volumeId* is in the list of storage objects, and check whether *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.<49>

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.<50>
TASK_INFO::clientID	Not required.<51>
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.<52>

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 **FT mirror set** on basic disks into two independent partitions. This is a synchronous task.<53>

```
HRESULT FTBreakMirror(
  [in] LdmObjectId volumeId,
  [in] hyper volumeLastKnownState,
  [in] boolean bForce,
```


Copyright © 2016 Microsoft Corporation

```
    [out] TASK_INFO* tinfo
  );
```

volumeId: Specifies the OID 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] section 2.1; 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 whether *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.<54>
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.<55>
TASK_INFO::clientID	Not required.<56>

Copyright © 2016 Microsoft Corporation

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.<57>

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 the partitions that results from the breakup.
4. Delete the file system object associated with the FT volume.
5. Create the file system objects associated with the partitions that result from the call to break the mirror.<58>

3.2.4.4.1.14 IVolumeClient::FTResyncMirror (Opnum 17)

The FTResyncMirror method restores the redundancy of an FT mirror set on basic disks by resynchronizing the members of the mirror. This is a synchronous task.<59>

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

volumeId: Specifies the OID of the FT mirror set that is 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] section 2.1; 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.

Copyright © 2016 Microsoft Corporation

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.<60>
TASK_INFO::clientID	Not required.<61>
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.

3.2.4.4.1.15 IVolumeClient::FTRegenerateParityStripe (Opnum 18)

The FTRegenerateParityStripe method restores the redundancy of an FT RAID-5 set on basic disks by regenerating the parity of the volume. This is a synchronous task.<63>

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

volumeId: Specifies the OID 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:

1. Verify that the FT volume specified by *volumeId* is in the list of storage objects, and check whether *volumeLastKnownState* matches the **LastKnownState** field of the object. Verify that the FT volume is an FT RAID-5 set.

2. Verify that *tinfo* is not NULL.

Copyright © 2016 Microsoft Corporation

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 parity regeneration for the FT volume specified by *volumeId*.

2. Wait for the parity regeneration 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.<64>
TASK_INFO::clientID	Not required.<65>
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.<66>

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

3.2.4.4.1.16 IVolumeClient::FTReplaceMirrorPartition (Opnum 19)

The FTReplaceMirrorPartition method repairs a FT mirror set on basic disks by replacing the failed member of the set with another partition.<67>

```
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 OID of the FT mirror set to modify.

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

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 OID of the replacement partition.

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

Copyright © 2016 Microsoft Corporation

flags: Bitmap of flags for the replacement operation. The value of this field is one of the applicable flags defined as follows.

Value	Meaning
FTREPLACE_FORCE 0x00000001	Do not fail the operation if the replacement partition has been changed since *newRegionLastKnownState*.
FTREPLACE_DELETE_ON_FAIL 0x00000002	Delete the replacement partition if the operation fails.
0x00000000	Fail the call if the input *newRegionLastKnownState* 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] section 2.1; 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 whether *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 whether *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.<68>
TASK_INFO::clientID	Not required.<69>
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.

Copyright © 2016 Microsoft Corporation

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.<70>

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 the 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 the 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 on basic disks by replacing the failed member of the set with another partition.<71>

```
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 OID 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.

Copyright © 2016 Microsoft Corporation

newRegionId: Specifies the OID 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 following applicable flags.

Value	Meaning
FTREPLACE_FORCE 0x00000001	Do not fail the operation if the replacement partition has been changed since *newRegionLastKnownState*.
FTREPLACE_DELETE_ON_FAIL 0x00000002	Delete the replacement partition if the 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] section 2.1; 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 whether *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 whether *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.<72>
TASK_INFO::clientID	Not required.<73>

Copyright © 2016 Microsoft Corporation

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.<74>

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 the 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 the 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.

3.2.4.4.1.18 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.

Copyright © 2016 Microsoft Corporation

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] section 2.1; 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 *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.<75>
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 preceding output parameters 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 *storageId* cannot be locked, the operation fails with LDM_E_VOLUME_IN_USE.
FORCE_OPERATION 0x00000001	If the volume, partition, or logical drive specified by *storageId* 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.

Copyright © 2016 Microsoft Corporation

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] section 2.1; 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 the letter is in the list of storage objects, and check whether *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 whether *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 uppercase.
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 path name associated with the volume. If the existing drive letter path name cannot be deleted, the server MUST fail the call.
5. Assign the drive 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.<76>
TASK_INFO::clientID	Not required.<77>
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>

Copyright © 2016 Microsoft Corporation

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] section 2.1; 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 whether *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 whether *storageLastKnownState* matches the **LastKnownState** field of the object.

Copyright © 2016 Microsoft Corporation

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*:<79>

   - 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.<80>
TASK_INFO::clientID	Not required.<81>
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.<82>

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 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.<83>

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


Copyright © 2016 Microsoft Corporation

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:

1. 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 previously 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:

1. 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.

Copyright © 2016 Microsoft Corporation

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 previously 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.<84>

quickFormat: Boolean value that indicates whether 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 whether 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.

Copyright © 2016 Microsoft Corporation

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 whether *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.<85>

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.<86>
TASK_INFO::clientID	Not required.<87>
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>

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.<89> 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.

Copyright © 2016 Microsoft Corporation

- 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 previously 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.<90>

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

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

memberCount: Pointer to the number of disk regions returned in *memberList*.

Copyright © 2016 Microsoft Corporation

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).<91>

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 previously 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).<92>

Upon receiving this message, the server MUST validate parameters:

- Verify that *diskCount* is not 0 and *diskList* is not NULL.

Copyright © 2016 Microsoft Corporation

- For each DISK_SPEC structure in *diskList*, verify that the disk specified by *diskId* is in the list of storage objects; and check whether *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 field layout and length of the *volumeSpec* parameter.
   - 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 noncontiguous disk regions.<93>
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 if the method succeeds.
TASK_INFO::createTime	Not required.<94>
TASK_INFO::clientID	Not required.<95>
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.<96>

4. Return a response to the client containing *tinfo* and the status of the operation.<97>
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.

Copyright © 2016 Microsoft Corporation

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.<98>

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 2-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 whether *lastKnownState* matches the **LastKnownState** field of the object.

Copyright © 2016 Microsoft Corporation

3. Verify that the drive letter object, if specified by letter, is in the list of storage objects, and check whether letterLastKnownState matches the **LastKnownState** field of the object. <99>

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 noncontiguous 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.<100>

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Required.
TASK_INFO::createTime	Not required.<101>
TASK_INFO::clientID	Not required.<102>
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.<103>

7. Return a response to the client that contains *tinfo* and the status of the operation.
8. Send the task completion notification.

Copyright © 2016 Microsoft Corporation

Note The server MAY decide not to wait for the formatting to complete before returning the response to the client.<104> 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 finished, 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 2-byte null character or the Unicode SPACE character.

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

Copyright © 2016 Microsoft Corporation

cchAccessPath: Length of *AccessPath*, including the terminating null character.

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 following applicable flags with a logical OR operation.

Value	Meaning
CREATE_ASSIGN_ACCESS_PATH 0x00000001	Assign the mount point *AccessPath* to the new volume. If the flag is not set, the parameter *AccessPath* is ignored.

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).

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 OID of the volume for which the mount name is being retrieved.

cchMountName: Pointer to the length of *mountName*, including the terminating null character.

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.

Copyright © 2016 Microsoft Corporation

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 the terminating null character.
3. Populate the buffer with the mount name, including the terminating null character.
4. The buffer MUST be returned to the client in the output parameter *mountName*.
5. The number of characters in the buffer, including the terminating null character, MUST be returned in the output parameter *cchMountName*.
6. Return the response that contains the preceding output parameters 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 OID 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.

Copyright © 2016 Microsoft Corporation

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).<105>

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 whether 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 whether **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 of the volume is determined by the field length 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.<106>
TASK_INFO::clientID	Not required.<107>
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.

Copyright © 2016 Microsoft Corporation

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.<108>

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 correspond 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.<109>

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 OID of the volume to delete.

force: A value that indicates whether deletion of the volume will be forced if the volume is in use by another application. 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).

Upon receiving this message, the server MUST validate parameters:

Copyright © 2016 Microsoft Corporation

1. Verify that the dynamic volume specified by *volumeId* is in the list of storage objects, and check whether **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.<110>
TASK_INFO::clientID	Not required.<111>
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.<112>

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.<113>

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.<114>

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,
```


Copyright © 2016 Microsoft Corporation

```
  [in] hyper volumeLastKnownState,
  [in] DISK_SPEC diskSpec,
  [in, out] int* diskNumber,
  [out] int* partitionNumber,
  [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the OID 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 the 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 whether 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 whether *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 a mirror.
   - The length of the member is determined by the length of the volume field.
   - 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 noncontiguous 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.

Copyright © 2016 Microsoft Corporation

TASK_INFO member	Required for this operation
TASK_INFO::id	Required<115>.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required.<116>
TASK_INFO::clientID	Not required.<117>
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.<118>

5. Return a response to the client containing *tinfo*, *partitionNumber*, and the status of the operation.

6. Send the task completion notification.<119>

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 OID 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.

Copyright © 2016 Microsoft Corporation

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 whether 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 whether *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.<120>
TASK_INFO::clientID	Not required.<121>
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.<122>

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 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.

Copyright © 2016 Microsoft Corporation

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 OID 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 2-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 whether 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 whether *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 whether *letterLastKnownState* matches the **LastKnownState** field of the object. <123>

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

Copyright © 2016 Microsoft Corporation

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 preceding sequence 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.<124>
TASK_INFO::clientID	Not required.<125>
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.<126>

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 the 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.<127>

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,
```


Copyright © 2016 Microsoft Corporation

```
    [in] hyper diskLastKnownState,
    [out] TASK_INFO* tinfo
  );
```

diskId: Specifies the OID 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 whether *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 MBR partition table and write an MBR signature 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.<128>
TASK_INFO::clientID	Not required.<129>
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.<130>

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:

Copyright © 2016 Microsoft Corporation

1. Modify the disk object to account for the change in type.
2. Delete disk region objects residing on the uninitialized disk.<131>
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 an asynchronous task.

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

diskId: Specifies the OID 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 whether *diskLastKnownState* matches the **LastKnownState** field of the object.
2. Verify that the disk specified by *diskId* is empty.<132>
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.<133>
TASK_INFO::clientID	Not required.<134>
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.
TASK_INFO::status	Required.

Copyright © 2016 Microsoft Corporation

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.<135>

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

5. Send the task completion notification.

Note The server MAY decide not to wait for the disk conversion to complete before returning the response to the client. This task is asynchronous and all rules for handling asynchronous tasks apply here.<136>

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.<137>

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 an asynchronous task.

```
HRESULT ReConnectDisk(
  [in] LdmObjectId diskId,
  [out] TASK_INFO* tinfo
);
```

diskId: Specifies the OID 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.

Copyright © 2016 Microsoft Corporation

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.<138>
TASK_INFO::clientID	Not required.<139>
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.<140>

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

5. Send the task completion notification.

Note The server MAY decide not to wait for the disk reactivation operation to complete before returning the response to the client. This task is asynchronous and all rules for handling asynchronous tasks apply here.<141>

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 region objects that reside on the disk to account for the change in status.

3. Modify the volume objects that reside on the disk to account for the change in status.

4. Modify drive letter objects to mark them as in use by the volumes brought online.

5. 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 an asynchronous task.

```
HRESULT ImportDiskGroup(
  [in] int cchDgid,
  [in, size_is(cchDgid)] byte* dgid,
  [out] TASK_INFO* tinfo
);
```

cchDgid: Size of *dgid* in characters, including the terminating null character.

dgid: Null-terminated string that contains the 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.

Copyright © 2016 Microsoft Corporation

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 foreign disk 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.<142>
TASK_INFO::clientID	Not required.<143>
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.<144>

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

5. Send the task completion notification.

Note The server MAY decide not to wait for the disk import operation to complete before returning the response to the client. This task is asynchronous and all rules for handling asynchronous tasks apply here.<145>

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.

Copyright © 2016 Microsoft Corporation

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 the terminating null character.

dgid: Null-terminated string containing the 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 OIDs 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 following applicable flags 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 *diskList* . 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.<146>

Copyright © 2016 Microsoft Corporation

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 preceding output parameters and the status of the operation.
14. Send the task completion notification.

The server MUST NOT change the list of storage objects as part of processing this message.<147>

3.2.4.4.1.40 IVolumeClient::DiskMerge (Opnum 45)

Copyright © 2016 Microsoft Corporation

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 the terminating null character.

dgid: Null-terminated string that contains the UUID of the disk group to be merged.

numDisks: Number of disks passed in *diskList*.

diskList: Array of OIDs of type LdmObjectId that specifies 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.<148>

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 foreign disk 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.

Copyright © 2016 Microsoft Corporation

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.<149>
TASK_INFO::clientID	Not required.<150>
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.<151>

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 foreign 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).

3.2.4.4.1.41 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.<152>

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

diskId: Specifies the OID 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).

Copyright © 2016 Microsoft Corporation

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 whether *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.<153>
TASK_INFO::clientID	Not required.<154>
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.<155>

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::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.

Copyright © 2016 Microsoft Corporation

```
HRESULT ReplaceRaid5Column(
  [in] LdmObjectId volumeId,
  [in] hyper volumeLastKnownState,
  [in] LdmObjectId newDiskId,
  [in] hyper diskLastKnownState,
  [out] TASK_INFO* tinfo
);
```

volumeId: Specifies the OID of the volume in which to replace the member.

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

newDiskId: Specifies the OID 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 whether *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 whether *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.<156>
TASK_INFO::clientID	Not required.<157>
TASK_INFO::percentComplete	Required for any task that returns REQ_IN_PROGRESS.

Copyright © 2016 Microsoft Corporation

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.<158>

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 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.

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 correspond 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.

3.2.4.4.1.43 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 OID 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 whether *volumeLastKnownState* matches the field **LastKnownState** of the object.

Copyright © 2016 Microsoft Corporation

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.<159>
TASK_INFO::clientID	Not required.<160>
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.<161>

4. Return a response to the client that contains *tinfo* and the status of the operation.<162>

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).

3.2.4.4.1.44 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,
```


Copyright © 2016 Microsoft Corporation

```
    [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 specifies 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 as follows 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 represents the disks that will be affected by the encapsulation. Memory for the array is allocated by the server and freed by the client.

Copyright © 2016 Microsoft Corporation

affectedDiskFlags: Pointer to an array of bitmaps of flags that provides 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 following applicable flags.

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 represents 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 represents 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.

Copyright © 2016 Microsoft Corporation

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 whether *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.<163>

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*. If disks that have existing FT Disk volume sets are being encapsulated, the server must get a list of volume **extents** for all volumes that have at least one extent on the input disks. Then walk through the list of volume extents and add it for each extent, if the disk on which the extent is located is not in the *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 *affectedDiskFlags*. Note that the number of elements in the buffer is the same as the count of disks, which is 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*.

Copyright © 2016 Microsoft Corporation

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, excluding free regions.

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 previously and the status of the operation.

22. Fill in the *tinfo* output parameter. This is a synchronous task.

    - The *tinfo* values MUST be set as follows.

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required.<164>
TASK_INFO::clientID	Not required.<165>
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.<166>

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

3.2.4.4.1.45 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,
```


Copyright © 2016 Microsoft Corporation

```
  [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 specifies the disks to be encapsulated.

affectedVolumeCount: The number of elements passed in the *affectedVolumeList* array.

affectedVolumeList: An array of VOLUME_INFO structures that represents the volumes affected by the encapsulation. If the number of affect volumes is zero, a pointer to a zero length array MUST be passed. This pointer MUST NOT be input as NULL.

affectedRegionCount: The number of elements passed in the *affectedRegionList* array.

affectedRegionList: An array of REGION_INFO structures that represents the regions affected by the encapsulation. If the number of affect regions is zero, a pointer to a zero length array MUST be passed. This pointer MUST NOT be input as NULL.

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 following applicable flags.

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.

Copyright © 2016 Microsoft Corporation

Value	Meaning
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.

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 *affectedDiskList* is not NULL and that *affectedDiskCount* is not 0.
2. For each DISK_INFO structure specified by *affectedDiskList*, verify that the disk specified by *diskId* is in the list of storage objects and that **lastKnownState** matches the **LastKnownState** field of the object.
3. For each DISK_INFO structure specified by *affectedDiskList*, verify that the disk specified by *diskId* is not a GPT disk.
4. For each REGION_INFO structure specified by *affectedRegionList*, verify that the region's style field does have the value PARTITIONSTYLE_GPT.
5. Verify that no other basic disks need to be encapsulated together with the disks specified by *affectedDiskList*.
6. Verify that *affectedVolumeList* is not NULL. If *affectedVolumeCount* is zero, a valid pointer to a zero-length array for the *affectVolumeList* MUST be passed in.
7. Verify that *affectedRegionList* is not NULL. If *affectedRegionCount* is zero, a valid pointer to a zero-length array for the *affectRegionList* MUST be passed in.
8. Verify that the list of basic volumes specified by *affectedVolumeList* matches the set of **basic volumes** that reside on the disks specified by *affectedDiskList*.
9. Verify that *encapInfoFlags* is not NULL.
10. 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.

Copyright © 2016 Microsoft Corporation

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required.<167>
TASK_INFO::clientID	Not required.<168>
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.<169>

5. Return a response to the client containing *tinfo* and the status of the operation.
6. Send the task completion notification.<170>

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.<171>
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 and if the partition number of the boot partition changes during the **disk encapsulation**, 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. The IVolumeClient::QueryChangePartitionNumbers method will return the original partition number and the new partition number. This information MAY be used to update boot settings if necessary.<172>

3.2.4.4.1.46 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.

Copyright © 2016 Microsoft Corporation

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 theDisk Management Remote Protocol).

Upon receiving this message, the server MUST validate parameters:

1. 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 previously and the status of the operation.

3.2.4.4.1.47 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).<173>
2. Wait for the deletion to succeed or fail.
3. Return a response to the client that contains the status of the operation.

3.2.4.4.1.48 IVolumeClient::SetDontShow (Opnum 57)

The SetDontShow method sets a Boolean value that indicates whether to show a disk initialization tool.<174>

```
HRESULT SetDontShow(
  [in] boolean bSetNoShow
);
```


Copyright © 2016 Microsoft Corporation

bSetNoShow: Boolean value that determines whether the New Disk Wizard is enabled or disabled.

Value	Meaning
FALSE 0	Enables 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 be displayed in the future.
TRUE 1	Disables 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 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.

3.2.4.4.1.49 IVolumeClient::GetDontShow (Opnum 58)

The GetDontShow method retrieves a Boolean value that indicates whether to show a disk initialization tool.<175>

```
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 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 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:

1. 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.

Copyright © 2016 Microsoft Corporation

2. The setting MUST be returned to the client in the output parameter *bGetNoShow*.

3. Return a response that contains the output parameters mentioned previously and the status of the operation.

3.2.4.4.1.50 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 previously 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.<176>

3.2.4.4.1.51 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
);
```


Copyright © 2016 Microsoft Corporation

id: Specifies the OID 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 a response that contains the output parameters mentioned previously 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::AbortTask (Opnum 69)

The AbortTask method aborts a task running on the server.

```
HRESULT AbortTask(
  [in] LdmObjectId id
);
```

id: Specifies the OID 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*. The 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.

Copyright © 2016 Microsoft Corporation

4. If successful, send task completion notification and delete the task object from the list of tasks currently running on the server.

3.2.4.4.1.53 IVolumeClient::HrGetErrorData (Opnum 70)

The HrGetErrorData method retrieves user-readable error information associated with an HRESULT error code.<177>

```
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 as follows, 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.<178>

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 is: "The request cannot be completed because the volume is open or in use. It MAY be configured as a system, boot, or page file 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:

Copyright © 2016 Microsoft Corporation

- Such error messages are produced by the server during failed client requests. Because 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.<179>

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 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 the terminating null character. 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. The 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 previously and the status of the operation.

3.2.4.4.1.54 IVolumeClient::Initialize (Opnum 71)

The Initialize method initializes the dialog 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.<180>

pdwFlags: Bitmap of information flags about the server. The value of this field is generated by combining zero or more of the following applicable flags with a logical OR operation.

Value	Meaning
SYSFLAG_SERVER 0x00000001	Server is running on Windows 2000 Server operating system and Windows Server 2003 operating system.

Copyright © 2016 Microsoft Corporation

Value	Meaning
SYSFLAG_ALPHA 0x00000002	Server is running on an Alpha processor.<181>
SYSFLAG_SYSPART_SECURE 0x00000004	System partition for the server is secure.<182>
SYSFLAG_NEC_98 0x00000008	Server is an NEC 98 computer, which supports assignment of drive letters A and B to partitions or volumes.<183>
SYSFLAG_LAPTOP 0x00000010	Server is a laptop and does not support dynamic disks.
SYSFLAG_WOLFPACK 0x00000020	Server is running on a Microsoft Cluster Server (MSCS) cluster.

clientId: Pointer to the client's OID.

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 MAY be ignored by the server.<184>

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 IDMNotify interface that is specified by *notificationInterface* in the **notifyInterface** field of the client object.<185>

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 *ulIDLVersion*.

Copyright © 2016 Microsoft Corporation

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.

3.2.4.4.1.55 IVolumeClient::Uninitialize (Opnum 72)

The Uninitialize method ends the dialog 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 SHOULD be ignored by the server.<186>

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:

    1. Dereference the pointer to the IDMNotify interface that is 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.

3.2.4.4.1.56 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:

Copyright © 2016 Microsoft Corporation

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 a response to the client that contains the status of the operation.

3.2.4.4.1.57 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.

3.2.4.4.1.58 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.

3.2.4.4.1.59 IVolumeClient::SecureSystemPartition (Opnum 76)

Copyright © 2016 Microsoft Corporation

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.<187>

```
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.

3.2.4.4.1.60 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.

3.2.4.4.1.61 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:

Copyright © 2016 Microsoft Corporation

- 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 previously and the status of the operation.

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

3.2.4.4.1.62 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 OID 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*.

Copyright © 2016 Microsoft Corporation

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 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 previously 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::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 the terminating null character.

path: Null-terminated mount point path to assign to the volume targeted (see mounted folder). This is Unicode.

targetId: Specifies the OID 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.

3.2.4.4.1.64 IVolumeClient::DeleteAccessPath (Opnum 81)

The DeleteAccessPath method deletes a specified mount point from a volume, a partition, or a logical drive.

Copyright © 2016 Microsoft Corporation

```
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 the terminating null character.

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 in the following sections, 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

Copyright © 2016 Microsoft Corporation

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 OID 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.<188>
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 previously 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 in the following sections, 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.<189>

Copyright © 2016 Microsoft Corporation

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
IVolumeClient3::FTEnumLogicalDiskMembers	Opnum: 14
IVolumeClient3::FTDeleteVolume	Opnum: 15
IVolumeClient3::FTBreakMirror	Opnum: 16
IVolumeClient3::FTResyncMirror	Opnum: 17
IVolumeClient3::FTRegenerateParityStripe	Opnum: 18
IVolumeClient3::FTReplaceMirrorPartition	Opnum: 19
IVolumeClient3::FTReplaceParityStripePartition	Opnum: 20
IVolumeClient3::EnumDriveLetters	Opnum: 21
IVolumeClient3::AssignDriveLetter	Opnum: 22
IVolumeClient3::FreeDriveLetter	Opnum: 23
IVolumeClient3::EnumLocalFileSystems	Opnum: 24
IVolumeClient3::GetInstalledFileSystems	Opnum: 25
IVolumeClient3::Format	Opnum: 26
IVolumeClient3::EnumVolumes	Opnum: 27
IVolumeClient3::EnumVolumeMembers	Opnum: 28
IVolumeClient3::CreateVolume	Opnum: 29
IVolumeClient3::CreateVolumeAssignAndFormat	Opnum: 30
IVolumeClient3::CreateVolumeAssignAndFormatEx	Opnum: 31
IVolumeClient3::GetVolumeMountName	Opnum: 32
IVolumeClient3::GrowVolume	Opnum: 33

Copyright © 2016 Microsoft Corporation

Method	Description
IVolumeClient3::DeleteVolume	Opnum: 34
IVolumeClient3::CreatePartitionsForVolume	Opnum: 35
IVolumeClient3::DeletePartitionsForVolume	Opnum: 36
IVolumeClient3::GetMaxAdjustedFreeSpace	Opnum: 37
IVolumeClient3::AddMirror	Opnum: 38
IVolumeClient3::RemoveMirror	Opnum: 39
IVolumeClient3::SplitMirror	Opnum: 40
IVolumeClient3::InitializeDiskEx	Opnum: 41
IVolumeClient3::UninitializeDisk	Opnum: 42
IVolumeClient3::ReConnectDisk	Opnum: 43
IVolumeClient3::ImportDiskGroup	Opnum: 44
IVolumeClient3::DiskMergeQuery	Opnum: 45
IVolumeClient3::DiskMerge	Opnum: 46
IVolumeClient3::ReAttachDisk	Opnum: 47
IVolumeClient3::ReplaceRaid5Column	Opnum: 48
IVolumeClient3::RestartVolume	Opnum: 49
IVolumeClient3::GetEncapsulateDiskInfoEx	Opnum: 50
IVolumeClient3::EncapsulateDiskEx	Opnum: 51
IVolumeClient3::QueryChangePartitionNumbers	Opnum: 52
IVolumeClient3::DeletePartitionNumberInfoFromRegistry	Opnum: 53
IVolumeClient3::SetDontShow	Opnum: 54
IVolumeClient3::GetDontShow	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
IVolumeClient3::EnumTasks	Opnum: 64
IVolumeClient3::GetTaskDetail	Opnum: 65

Copyright © 2016 Microsoft Corporation

Method	Description
IVolumeClient3::AbortTask	Opnum: 66
IVolumeClient3::HrGetErrorData	Opnum: 67
IVolumeClient3::Initialize	Opnum: 68
IVolumeClient3::Uninitialize	Opnum: 69
IVolumeClient3::Refresh	Opnum: 70
IVolumeClient3::RescanDisks	Opnum: 71
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.

Copyright © 2016 Microsoft Corporation

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 previously 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 OID 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 describes 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 previously and the status of the operation.

Copyright © 2016 Microsoft Corporation

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.<190>

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.

Copyright © 2016 Microsoft Corporation

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 the terminating null character.

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 *AccessPath* to the new partition.

Copyright © 2016 Microsoft Corporation

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 OID of the target disk for the signature.

style: Value from the PARTITIONSTYLE enumeration that indicates the partition style to use.

Copyright © 2016 Microsoft Corporation

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 whether *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.<191>
TASK_INFO::clientID	Not required.<192>
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.<193>

4. 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 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.

Copyright © 2016 Microsoft Corporation

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

regionId: Specifies the OID 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 OID 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 FT volumes on basic disks (rather than dynamic disks).<194>

```
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.

Copyright © 2016 Microsoft Corporation

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 FT volume on basic disks (rather than dynamic disks).<195>

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

volumeId: Specifies the OID of the volume for which the 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 FT volume specified by *volumeId* on basic disks (rather than dynamic disks).<196>

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

volumeId: Specifies the OID 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.

Copyright © 2016 Microsoft Corporation

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 IVolumeClient::FTDeleteVolume (section 3.2.4.4.1.12).

3.2.4.4.3.13 IVolumeClient3::FTBreakMirror (Opnum 16)

The FTBreakMirror method breaks a specified FT mirror set on basic disks into two independent partitions.<197>

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

volumeId: Specifies the OID 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 on basic disks by resynchronizing the members of the mirror.<198>

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


Copyright © 2016 Microsoft Corporation

volumeId: Specifies the OID 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 on basic disks by regenerating the parity of the volume.<199>

```
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 an FT volume on basic disks (rather than dynamic disks). 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.<200>

```
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 OID of the FT mirror set to modify.

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

Copyright © 2016 Microsoft Corporation

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 OID 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 following applicable flags.

Value	Meaning
FTREPLACE_FORCE 0x00000001	Do not fail the operation if the replacement partition has been changed since *newRegionLastKnownState*.
FTREPLACE_DELETE_ON_FAIL 0x00000002	Delete the replacement partition if the 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 IVolumeClient::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 OID 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 OID 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.

Copyright © 2016 Microsoft Corporation

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 *newRegionLastKnownState*.
FTREPLACE_DELETE_ON_FAIL 0x00000002	Delete the replacement partition if the 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
);
```


Copyright © 2016 Microsoft Corporation

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 0x00000000	If the volume, partition, or logical drive specified by *storageId* already has a drive letter assigned, and freeing it fails because the object is in use, assignment fails and the old drive letter is retained.
FORCE_OPERATION 0x00000001	If the volume, partition, or logical drive specified by *storageId* 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 IVolumeClient::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.<201>

```
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.

Copyright © 2016 Microsoft Corporation

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).

Copyright © 2016 Microsoft Corporation

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.

Copyright © 2016 Microsoft Corporation

```
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 OID of the volume for which the 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.

Copyright © 2016 Microsoft Corporation

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 IVolumeClient::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 2-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)

Copyright © 2016 Microsoft Corporation

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 the terminating null character.

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 following applicable flags with a logical OR operation.

Value	Meaning
CREATE_ASSIGN_ACCESS_PATH 0x00000001	Assign the mount point *AccessPath* to the new volume. If the flag is not set, the parameter *AccessPath* is ignored.

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).

Copyright © 2016 Microsoft Corporation

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 OID of the volume for which the mount name is being retrieved.

cchMountName: Pointer to the length of *mountName*, including the terminating null character.

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 OID 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	Volume is not extended unless it is locked.

Copyright © 2016 Microsoft Corporation

Value	Meaning
0	
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 OID of the volume to delete.

force: A value that indicates whether deletion of the volume will be forced if the volume is in use by another application. 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(
```


Copyright © 2016 Microsoft Corporation

```
    [in] LdmObjectId volumeId,
    [in] boolean active,
    [in] hyper volumeLastKnownState,
    [out] TASK_INFO* tinfo
  );
```

volumeId: Specifies the OID of the volume under which to create a partition.

active: Boolean value that indicates whether 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 whether 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.<202>

Copyright © 2016 Microsoft Corporation

TASK_INFO member	Required for this operation
TASK_INFO::clientID	Not required.<203>
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.<204>

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

6. Send the task completion notification.<205>

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.<206>

3.2.4.4.3.33 IVolumeClient3::DeletePartitionsForVolume (Opnum 36)

The DeletePartitionsForVolume method deletes the partitions underneath a dynamic disk volume. This is a synchronous task.

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

volumeId: Specifies the OID 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 whether 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:

[MS-DMRP-Diff] - v20160714
Disk Management Remote Protocol
Copyright © 2016 Microsoft Corporation
Release: July 14, 2016

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.<207>
TASK_INFO::clientID	Not required.<208>
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.<209>

5. Return a response to the client containing *volumeId* and the status of the operation.

6. Send the task completion notification.<210>

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 OID 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).

Copyright © 2016 Microsoft Corporation

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 OID 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 OID of the mirrored volume from which the disk is being removed.

volumeLastKnownState: Volume's last known modification sequence number.

diskId: Specifies the OID 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.

Copyright © 2016 Microsoft Corporation

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 the volumes keeps the identifier and drive letter of the original volume. The other volume 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 OID of the volume to split.

volumeLastKnownState: Volume's last known modification sequence number.

diskId: Specifies the OID of the disk to split 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 2-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
);
```


Copyright © 2016 Microsoft Corporation

diskId: Specifies the OID of the disk to initialize for volume manager control.

style: Value from the PARTITIONSTYLE enumeration, which 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 whether *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.<211>
TASK_INFO::clientID	Not required.<212>
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.<213>

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

6. Send the task completion notification.

Copyright © 2016 Microsoft Corporation

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.<214>
- 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 OID 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 OID 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.

Copyright © 2016 Microsoft Corporation

```
HRESULT ImportDiskGroup(
  [in] int cchDgid,
  [in, size_is(cchDgid)] byte* dgid,
  [out] TASK_INFO* tinfo
);
```

cchDgid: Size of *dgid* in characters, including the terminating null character.

dgid: Null-terminated ASCII string that contains the 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 the terminating null character.

dgid: Null-terminated ASCII string that contains the UUID of the disk group to be merged.

numDisks: Number of disks passed in *diskList*.

diskList: Array of OIDs 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 following applicable flags.

Copyright © 2016 Microsoft Corporation

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 *diskList*. 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 IVolumeClient::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 the terminating null character.

dgid: Null-terminated ASCII string that contains the 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)

Copyright © 2016 Microsoft Corporation

The ReAttachDisk method reattaches the specified dynamic disk, bringing the volumes of the disk online after reconnecting the disk device to the server.<215>

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

diskId: Specifies the OID of the disk to reattach.

diskLastKnownState: The 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 OID of the volume for which the member will be replaced.

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

newDiskId: Specifies the OID of the replacement disk.

diskLastKnownState: Last known modification sequence number of the replacement 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::ReplaceRaid5Column message, it MUST process that message, as specified in IVolumeClient::ReplaceRaid5Column (section 3.2.4.4.1.42).

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,
```


Copyright © 2016 Microsoft Corporation

```
    [out] TASK_INFO* tinfo
  );
```

volumeId: Specifies the OID of the volume to restart.

volumeLastKnownState: Last known modification sequence number of the volume.

tinfo: 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).

When the server receives an IVolumeClient3::RestartVolume message, it MUST process that message, as specified in IVolumeClient::RestartVolume (section 3.2.4.4.1.43).

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 following applicable flags 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	Volume manager was unable to determine the type of a partition on the

Copyright © 2016 Microsoft Corporation

Value	Meaning
0x00000008	disk because of corruption or other errors reading the disk. For example, any error that prevents the partition information from being read, or the partition is neither GPT nor MBR, or 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 nonbasic 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 following applicable flags 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.

Copyright © 2016 Microsoft Corporation

Value	Meaning
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 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 *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 whether *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*.

Copyright © 2016 Microsoft Corporation

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 number of elements in the buffer is the same as the count of disks, which is 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 previously 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)

Copyright © 2016 Microsoft Corporation

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 specifies the disks to be encapsulated.

affectedVolumeCount: The number of elements passed in the *affectedVolumeList* array.

affectedVolumeList: An array of VOLUME_INFO structures that represents the volumes affected by the encapsulation. If the number of affect volumes is zero, a pointer to a zero-length array MUST be passed. This pointer MUST NOT be input as NULL.

affectedRegionCount: The number of elements passed in the *affectedRegionList* array.

affectedRegionList: An array of REGION_INFO_EX structures that represents the regions affected by the encapsulation. If the number of affect regions is zero, a pointer to a zero-length array MUST be passed. This pointer MUST NOT be input as NULL.

encapInfoFlags: Bitmap of flags that provides information about the encapsulation. The value of this field is generated by combining zero or more of the following applicable flags with a logical OR operation.

Value	Meaning
ENCAP_INFO_CANT_PROCEED 0x00000001	Encapsulation for disk did not succeed. Inspect the other values of *encapInfoFlags* 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.

Copyright © 2016 Microsoft Corporation

Value	Meaning
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 GPT disk that contains basic partitions mixed with nonbasic 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 that **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 not NULL. If *affectedVolumeCount* is zero, a valid pointer to a zero-length array for the *affectVolumeList* MUST be passed in.
5. Verify that *affectedRegionList* is not NULL. If *affectedRegionCount* is zero, a valid pointer to a zero-length array for the *affectRegionList* MUST be passed in.
6. Verify that the list of basic volumes specified by *affectedVolumeList* matches the set of basic volumes residing on the disks specified by *affectedDiskList*.
7. Verify that *encapInfoFlags* is not NULL.
8. 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.

Copyright © 2016 Microsoft Corporation

TASK_INFO member	Required for this operation
TASK_INFO::id	Required.
TASK_INFO::storageId	Not required.
TASK_INFO::createTime	Not required.<216>
TASK_INFO::clientID	Not required.<217>
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.<218>

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

5. Send the task completion notification.<219>

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.<220>

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.

Copyright © 2016 Microsoft Corporation

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 3.2.4.4.1.46).

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 3.2.4.4.1.47).

3.2.4.4.3.51 IVolumeClient3::SetDontShow (Opnum 54)

The SetDontShow method sets a Boolean value that indicates whether to show a disk initialization tool. <221>

```
HRESULT SetDontShow(
  [in] boolean bSetNoShow
);
```

bSetNoShow: Boolean value that determines whether the New Disk Wizard is enabled.

Value	Meaning
FALSE 0	Enables the 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 be displayed in the future.
TRUE 1	Disables the 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 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 3.2.4.4.1.48).

3.2.4.4.3.52 IVolumeClient3::GetDontShow (Opnum 55)

The GetDontShow method retrieves a value that indicates whether to show a disk initialization tool. <222>

Copyright © 2016 Microsoft Corporation

```
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 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 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 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 3.2.4.4.1.49).

3.2.4.4.3.53 IVolumeClient3::EnumTasks (Opnum 64)

The EnumTasks method enumerates the tasks that are 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 3.2.4.4.1.50).

3.2.4.4.3.54 IVolumeClient3::GetTaskDetail (Opnum 65)

The GetTaskDetail method retrieves information about a task that is running on the server.

```
HRESULT GetTaskDetail(
  [in] LdmObjectId id,
  [in, out] TASK_INFO* tinfo
);
```

id: Specifies the OID of the task for which to retrieve information.

tinfo: A TASK_INFO structure that describes the operation currently being performed by *id*.

Copyright © 2016 Microsoft Corporation

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 3.2.4.4.1.51).

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 OID 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 3.2.4.4.1.52).

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 following applicable flags 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.<223>

pcszw: Pointer to the number of strings returned in *prgszw*.

prgszw: Pointer to an array of strings that contain error information for the HRESULT.

Copyright © 2016 Microsoft Corporation

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 3.2.4.4.1.53).

3.2.4.4.3.57 IVolumeClient3::Initialize (Opnum 68)

The Initialize method initializes the dialog 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 that is 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 following applicable flags with a logical OR operation.

Value	Meaning
SYSFLAG_SERVER 0x00000001	Server is running on Windows 2000 Server and Windows Server 2003.
SYSFLAG_ALPHA 0x00000002	Server is running on an Alpha processor.<224>
SYSFLAG_SYSPART_SECURE 0x00000004	System partition for the server is secure.<225>
SYSFLAG_NEC_98 0x00000008	Server is an NEC 98 computer, which supports assignment of drive letters A and B to partitions or volumes.<226>
SYSFLAG_NO_DYNAMIC 0x00000010	Server is a laptop and does not support dynamic disks.
SYSFLAG_WOLFPACK 0x00000020	Server is running on an MCS cluster.
SYSFLAG_IA64 0x00000040	Server is running on an Intel Itanium-based processor.
SYSFLAG_UNINSTALL_VALID 0x00000080	Server has an available and valid backup for uninstallation.
SYSFLAG_DYNAMIC_1394 0x00000100	Server supports converting IEEE 1394 attached disks to dynamic disks.

clientId: Pointer to the client's OID.

Copyright © 2016 Microsoft Corporation

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 3.2.4.4.1.54).

3.2.4.4.3.58 IVolumeClient3::Uninitialize (Opnum 69)

The Uninitialize method ends the dialog 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 3.2.4.4.1.55).

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 3.2.4.4.1.56).

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).

Copyright © 2016 Microsoft Corporation

When the server receives an IVolumeClient3::RescanDisks message, it MUST process that message, as specified in IVolumeClient::RescanDisks (section 3.2.4.4.1.57).

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 3.2.4.4.1.58).

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 3.2.4.4.1.59).

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 3.2.4.4.1.60).

3.2.4.4.3.64 IVolumeClient3::EnumAccessPath (Opnum 75)

The EnumAccessPath method enumerates all mount points configured on the machine.

```
HRESULT EnumAccessPath(
```


Copyright © 2016 Microsoft Corporation

```
  [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 3.2.4.4.1.61).

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] LdmObjectId VolumeId,
  [in, out] int* lCount,
  [out, size_is(,*lCount)] COUNTED_STRING** paths
);
```

volumeId: Specifies the OID 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 3.2.4.4.1.62).

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 the terminating null character.

path: Null-terminated mount point path to assign to the volume *targetId*.

targetId: Specifies the OID of the volume, partition, or logical drive to which the new mount point is to be assigned.

Copyright © 2016 Microsoft Corporation

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 3.2.4.4.1.63).

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 OID of the volume, partition, or logical drive from which to delete the mount point.

cch_path: Length of path in characters, including the terminating null character.

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 3.2.4.4.1.64).

3.2.4.4.4 IVolumeClient4

Unless otherwise specified in the following sections, 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.<227>

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.

Copyright © 2016 Microsoft Corporation

```
HRESULT RefreshEx(void);
```

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.<228>

In addition to the preceding actions, the server MUST check whether the lengths 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 operating system 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 OID of the volume whose path name is being returned.

cchVolumeDevice: Number of characters returned in *pwszVolumeDevice*, including the terminating null character.

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 the terminating null character.

Copyright © 2016 Microsoft Corporation

3. Populate the buffer with the device name, including the terminating null character.
4. The buffer MUST be returned to the client in the output parameter *pwszVolumeDevice*.
5. The number of characters in the buffer, including the terminating null character, MUST be returned in the output parameter *cchVolumeDevice*.
6. Return a response that contains the preceding output parameters 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 the 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 the terminating null character.

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") 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 to receive notifications from the operating system related to changes in the storage configuration of the system. Examples of causes of such changes include administrative change of the hardware configuration, hardware failures, and administrative configuration of storage objects using various tools.

Copyright © 2016 Microsoft Corporation

Note Failure to register for and handle storage change notifications impairs the capability of clients to perform configuration operations in the situations enumerated above .<229>

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** field 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 reenumerate 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.

Copyright © 2016 Microsoft Corporation

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.

Copyright © 2016 Microsoft Corporation

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.

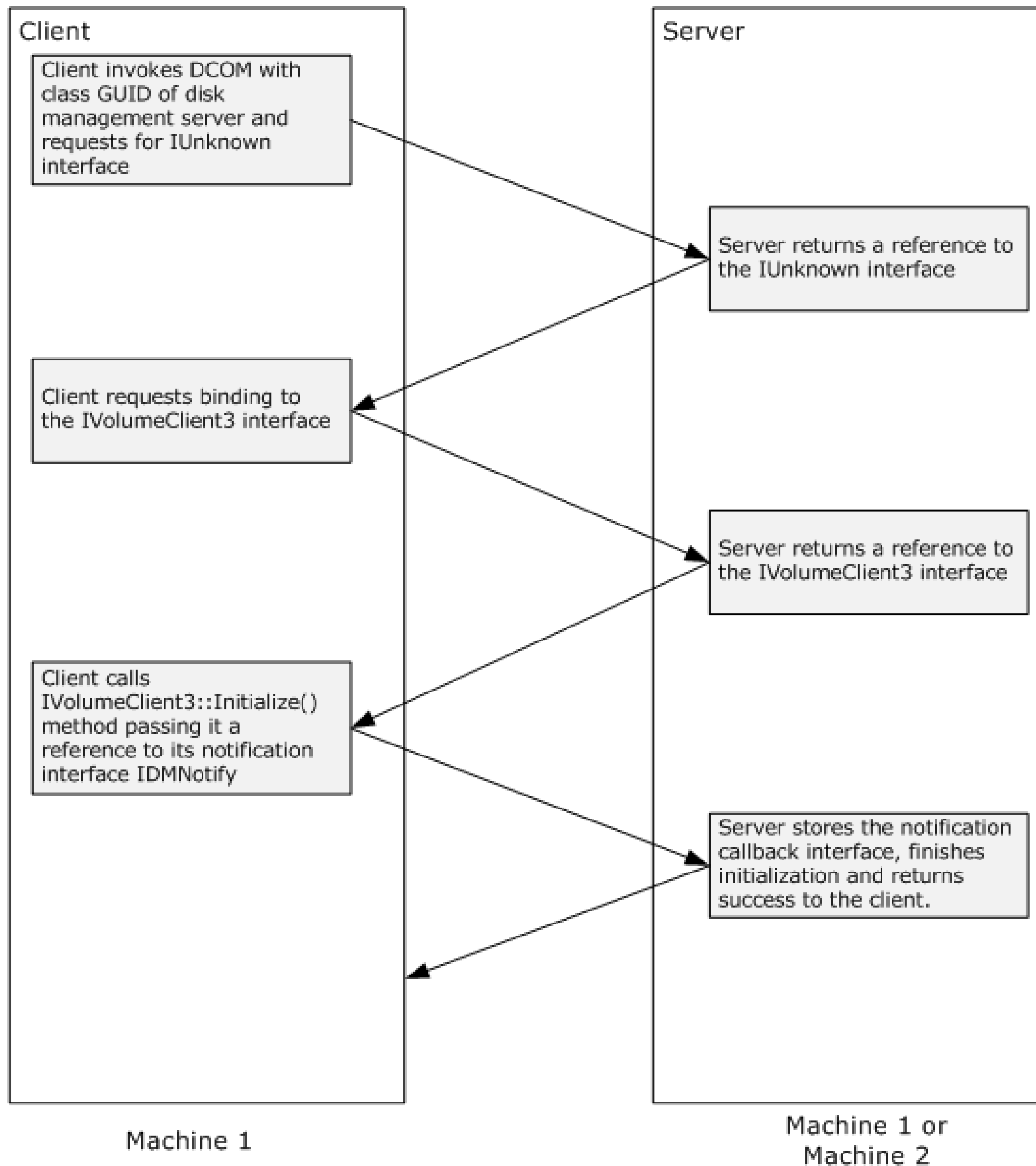

Figure 1: Steps to start a new session on a local or remote server

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.

Copyright © 2016 Microsoft Corporation

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 a remote server by using the IDMRemoteServer interface.

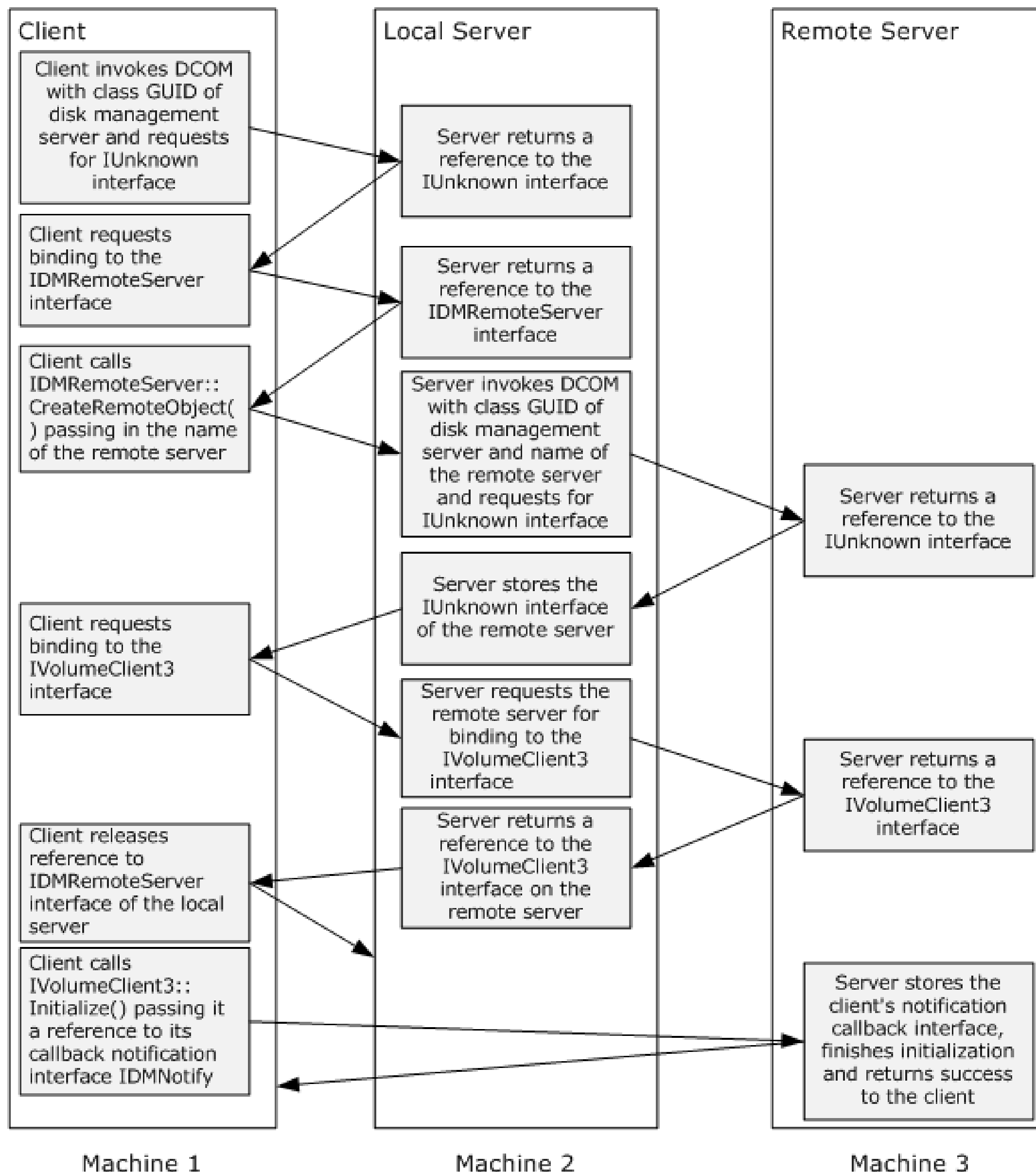

Figure 2: Steps to start a new session on a remote server using IDmRemoteServer interface

Copyright © 2016 Microsoft Corporation

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 the 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 the 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 the 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 in to the remote server by 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~~can now release the IDmRemoteServer interface, because 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.

Copyright © 2016 Microsoft Corporation

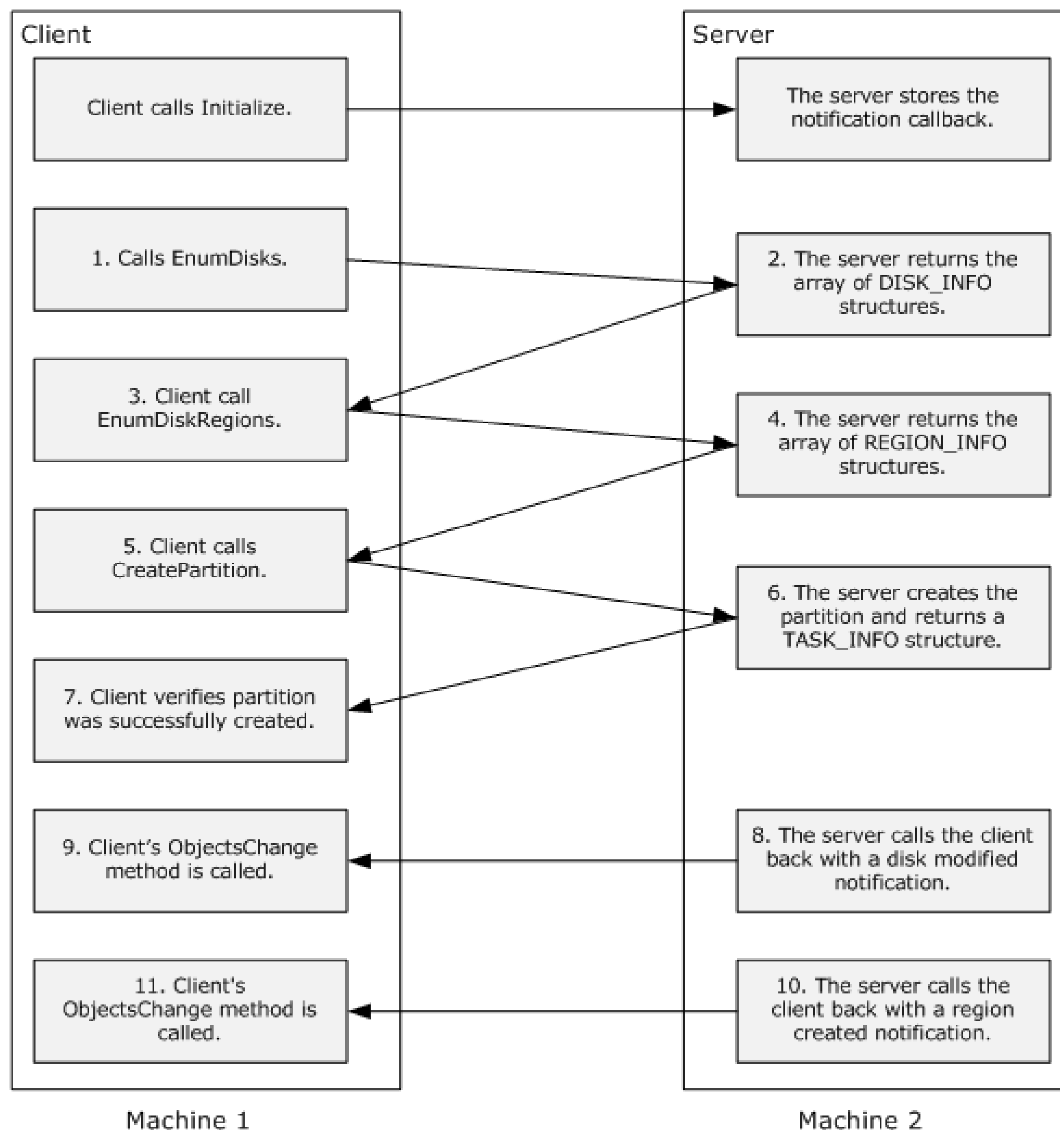

Figure 3: Steps to create a partition on a disk by using the IVolumeClient interfaces

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.

Copyright © 2016 Microsoft Corporation

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 mustcannot be no greater than the reported region length from the REGION_INFO structure. The start mustis required to 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 maymight 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 created notification on the IDMNotify interface's ObjectsChanged method.

12. The client code processes the region created notification. For example, the client maymight query for all disk regions 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.

Copyright © 2016 Microsoft Corporation

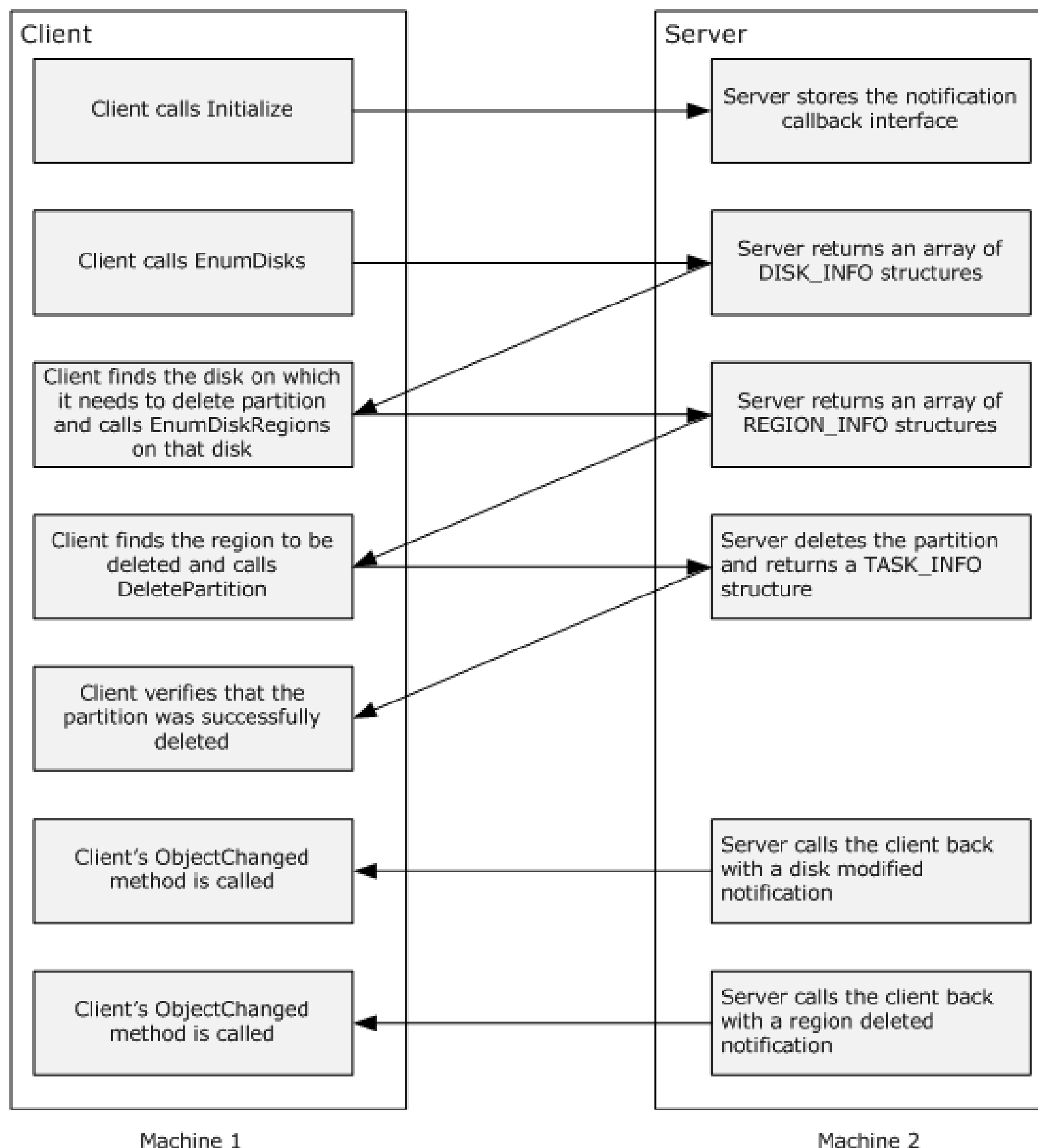

Figure 4: Steps to delete a partition on a disk by using the IVolumeClient interfaces

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 that 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.

Copyright © 2016 Microsoft Corporation

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 that 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 IDMNotify Interface's ObjectsChanged method.

10. The client code processes the disk modified notification. For example, the client ~~may~~might 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 IDMNotify Interface's ObjectsChanged method.

12. The client code processes the region deleted notification. For example, the client ~~may~~might query for all disk regions 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.

Copyright © 2016 Microsoft Corporation

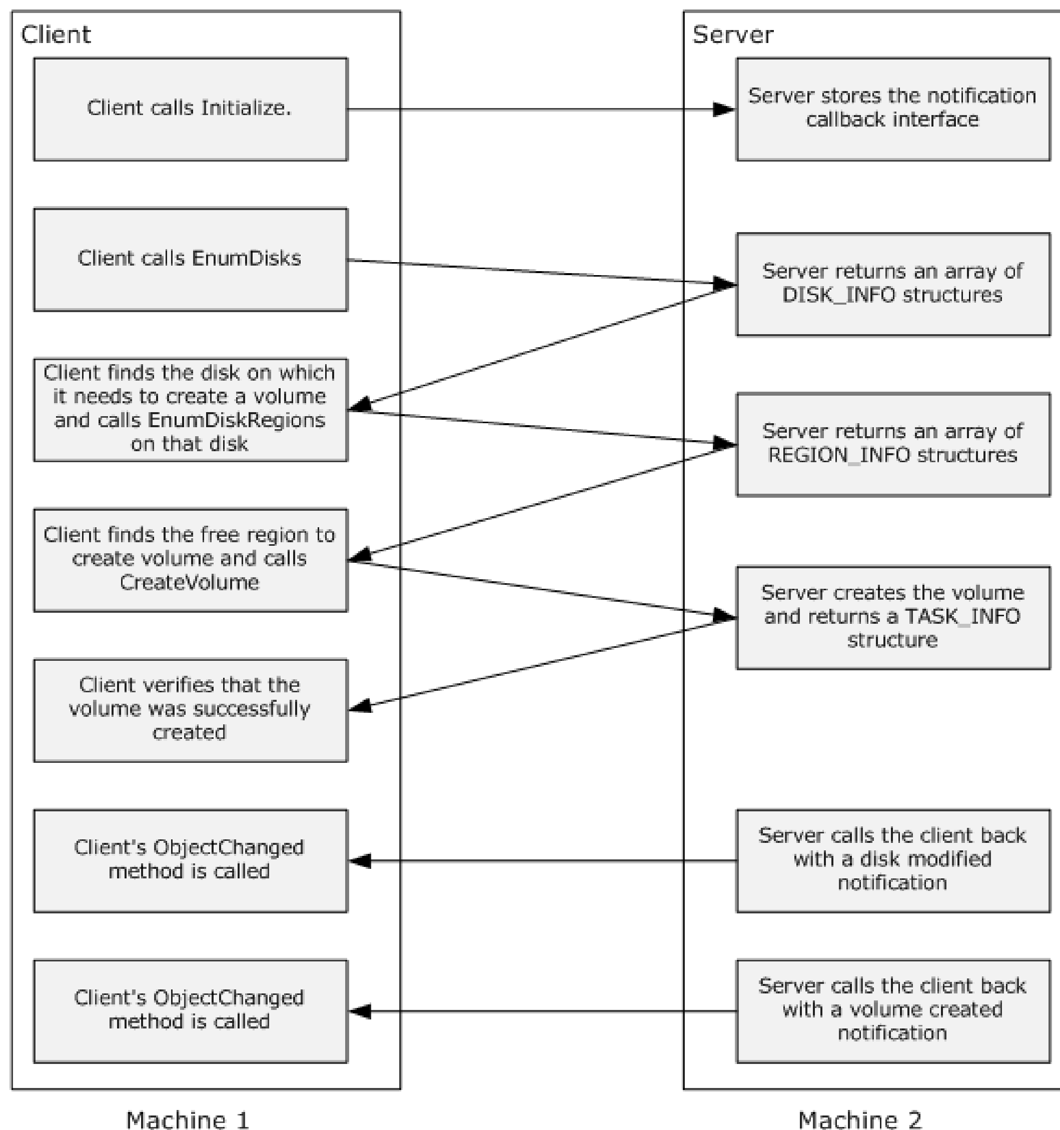

Figure 5: Steps to create a volume on a disk by using the IVolumeClient interfaces

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.

Copyright © 2016 Microsoft Corporation

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 disk **id** 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~~cannot 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~~might 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~~might 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.

Copyright © 2016 Microsoft Corporation

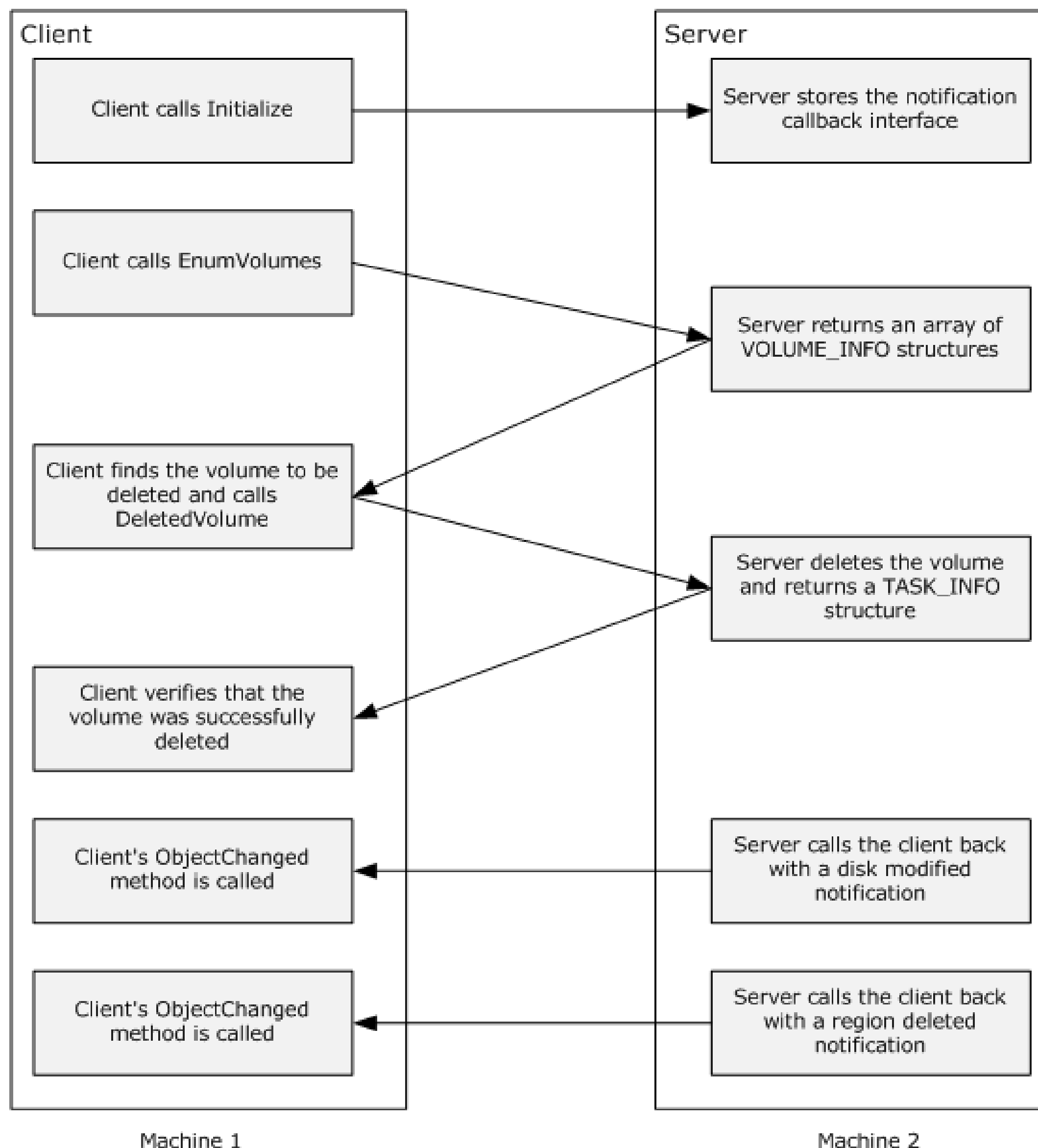

Figure 6: Steps to delete a volume on a disk by using the IVolumeClient interfaces

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

Copyright © 2016 Microsoft Corporation

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~~might 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~~might query for all volumes when it gets a volume deleted notification, so that it can update its cache or display.

Copyright © 2016 Microsoft Corporation

5 Security Considerations

This protocol introduces no security considerations beyond those applicable to DCOM interfaces, as specified in [MS-DCOM] section 5.<230>

Note For IDMNotify implementations, the DMRP server calls back into the client to setup the callback connection; in this instance the client implementing the IDMNotify interface acts like a server, and the DMRP server acts like a client; therefore the normal security considerations for a client to connect to a **COM** server ~~must~~are to be followed in this case.

Note For restrictions on remote anonymous calls, refer to [MS-RPCE] section 3.1.1.5.4.

Copyright © 2016 Microsoft Corporation

6 Appendix A: Full IDL

6.1 Appendix A.1: dmintf.idl

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

typedef LONGLONG LdmObjectId;

typedef enum _REGIONTYPE { REGION_UNKNOWN,
                           REGION_FREE,
                           REGION_EXTENDED_FREE,
                           REGION_PRIMARY,
                           REGION_LOGICAL,
                           REGION_EXTENDED,
                           REGION_SUBDISK,
                           REGION_CDROM,
                           REGION_REMOVABLE
                          } REGIONTYPE;

typedef enum _VOLUMETYPE {  VOLUMETYPE_UNKNOWN,
                            VOLUMETYPE_PRIMARY_PARTITION,
                            VOLUMETYPE_LOGICAL_DRIVE,
                            VOLUMETYPE_FT,
                            VOLUMETYPE_VM,
                            VOLUMETYPE_CDROM,
                            VOLUMETYPE_REMOVABLE
                          } VOLUMETYPE;

typedef enum _VOLUMELAYOUT { VOLUMELAYOUT_UNKNOWN,
                             VOLUMELAYOUT_PARTITION,
                             VOLUMELAYOUT_SIMPLE,
                             VOLUMELAYOUT_SPANNED,
                             VOLUMELAYOUT_MIRROR, VOLUMELAYOUT_STRIPE,
                             VOLUMELAYOUT_RAID5
                            } VOLUMELAYOUT;

typedef enum _REQSTATUS { REQ_UNKNOWN,
                          REQ_STARTED,
                          REQ_IN_PROGRESS,
                          REQ_COMPLETED,
                          REQ_ABORTED,
                          REQ_FAILED
                         } REQSTATUS;

typedef enum _REGIONSTATUS { REGIONSTATUS_UNKNOWN,
                             REGIONSTATUS_OK,
                             REGIONSTATUS_FAILED,
                             REGIONSTATUS_FAILING,
                             REGIONSTATUS_REGENERATING,
                             REGIONSTATUS_NEEDSRESYNC
                            } REGIONSTATUS;

typedef enum _VOLUMESTATUS {
                            VOLUME_STATUS_UNKNOWN,
                            VOLUME_STATUS_HEALTHY,
                            VOLUME_STATUS_FAILED,
                            VOLUME_STATUS_FAILED_REDUNDANCY,
                            VOLUME_STATUS_FAILING,
                            VOLUME_STATUS_FAILING_REDUNDANCY,
                            VOLUME_STATUS_FAILED_REDUNDANCY_FAILING,
                            VOLUME_STATUS_SYNCHING,
                            VOLUME_STATUS_REGENERATING,
                            VOLUME_STATUS_INITIALIZING,
                            VOLUME_STATUS_FORMATTING
                            } VOLUMESTATUS;
```


Copyright © 2016 Microsoft Corporation

```
typedef enum _LDMACTION { LDMACTION_UNKNOWN,
                          LDMACTION_CREATED,
                          LDMACTION_DELETED,
                          LDMACTION_MODIFIED,
                          LDMACTION_FAILED
                         } LDMACTION;

typedef enum _dmNotifyInfoType { DMNOTIFY_UNKNOWN_INFO,
                                 DMNOTIFY_DISK_INFO,
                                 DMNOTIFY_VOLUME_INFO,
                                 DMNOTIFY_REGION_INFO,
                                 DMNOTIFY_TASK_INFO,
                                 DMNOTIFY_DL_INFO,
                                 DMNOTIFY_FS_INFO,
                                 DMNOTIFY_SYSTEM_INFO } DMNOTIFY_INFO_TYPE;


typedef enum _dmProgressType { PROGRESS_UNKNOWN,
                               PROGRESS_FORMAT,
                               PROGRESS_SYNCHING } DMPROGRESS_TYPE;

const DWORD DISK_AUDIO_CD             = 0x1;
const DWORD DISK_NEC98                = 0x2;

#define DEVICETYPE_UNKNOWN        0x00000000
#define DEVICETYPE_VM                    0x00000001
#define DEVICETYPE_REMOVABLE      0x00000002
#define DEVICETYPE_CDROM          0x00000003
#define DEVICETYPE_FDISK          0x00000004
#define DEVICETYPE_DVD                   0x00000005

#define DEVICESTATE_UNKNOWN                     0x00000000
#define DEVICESTATE_HEALTHY                     0x00000001
#define DEVICESTATE_NO_MEDIA             0x00000002
#define DEVICESTATE_NOSIG                0x00000004
#define DEVICESTATE_BAD                         0x00000008
#define DEVICESTATE_NOT_READY            0x00000010
#define DEVICESTATE_MISSING                     0x00000020
#define DEVICESTATE_OFFLINE                     0x00000040
#define DEVICESTATE_FAILING                     0x00000080
#define DEVICESTATE_IMPORT_FAILED 0x00000100
#define DEVICESTATE_UNCLAIMED            0x00000200

#define BUSTYPE_UNKNOWN   0x00000000
#define BUSTYPE_IDE              0x00000001
#define BUSTYPE_SCSI      0x00000002
#define BUSTYPE_FIBRE     0x00000003
#define BUSTYPE_USB              0x00000004
#define BUSTYPE_SSA              0x00000005
#define BUSTYPE_1394      0x00000006

#define DEVICEATTR_NONE          0x00000000
#define DEVICEATTR_RDONLY 0x00000001
#define DEVICEATTR_NTMS          0x00000002

#define CONTAINS_FT                                    0x00000001
#define CONTAINS_RAID5                          0x00000002
#define CONTAINS_REDISTRIBUTION          0x00000004
#define CONTAINS_BOOTABLE_PARTITION      0x00000008


#define CONTAINS_LOCKED_PARTITION 0x00000010
#define CONTAINS_NO_FREE_SPACE           0x00000020
#define CONTAINS_EXTENDED_PARTITION      0x00000040
#define PARTITION_NUMBER_CHANGE          0x00000080
#define CONTAINS_BOOTINDICATOR           0x00000100
#define CONTAINS_BOOTLOADER                     0x00000200
#define CONTAINS_SYSTEMDIR               0x00000400
#define CONTAINS_MIXED_PARTITIONS 0x00000800
```


Copyright © 2016 Microsoft Corporation

```
const unsigned long PARTITION_OS2_BOOT   = 0xa;
const unsigned long       PARTITION_EISA                          = 0x12;
const unsigned long PARTITION_HIBERNATION= 0x84;
const unsigned long PARTITION_DIAGNOSTIC = 0xA0;
const unsigned long PARTITION_DELL                                = 0xDE;
const unsigned long PARTITION_IBM                          = 0xFE;


const DWORD REGION_FORMAT_IN_PROGRESS   = 0x1;
const DWORD VOLUME_FORMAT_IN_PROGRESS   = 0x1;
const DWORD REGION_IS_SYSTEM_PARTITION  = 0x2;
const DWORD REGION_HAS_PAGEFILE         = 0x4;
const DWORD VOLUME_HAS_PAGEFILE         = 0x4;
const DWORD REGION_HAD_BOOT_INI         = 0x40;
const DWORD VOLUME_IS_BOOT_VOLUME       = 0x100;
const DWORD VOLUME_IS_RESTARTABLE       = 0x400;
const DWORD VOLUME_IS_SYSTEM_VOLUME     = 0x800;
const DWORD VOLUME_HAS_RETAIN_PARTITION = 0x1000;
const DWORD VOLUME_HAD_BOOT_INI         = 0x2000;
const DWORD VOLUME_CORRUPT              = 0x4000;
const DWORD VOLUME_HAS_CRASHDUMP        = 0x8000;
const DWORD VOLUME_IS_CURR_BOOT_VOLUME  = 0x10000;
const DWORD VOLUME_HAS_HIBERNATION      = 0x20000;

const DWORD NO_FORCE_OPERATION          = 0;
const DWORD FORCE_OPERATION             = 1;

const DWORD DL_PENDING_REMOVAL          = 0x1;

const DWORD SYSFLAG_SERVER              = 0x1;
const DWORD SYSFLAG_ALPHA               = 0x2;
const DWORD SYSFLAG_SYSPART_SECURE      = 0x4;
const DWORD SYSFLAG_NEC_98              = 0x8;
const DWORD SYSFLAG_LAPTOP              = 0x10;
const DWORD SYSFLAG_WOLFPACK            = 0x20;

const DWORD DSKMERGE_DELETE             = 0x1;
const DWORD DSKMERGE_DELETE_REDUNDANCY  = 0x2;
const DWORD DSKMERGE_STALE_DATA         = 0x4;
const DWORD DSKMERGE_RELATED            = 0x8;

const DWORD DSKMERGE_IN_NO_UNRELATED    = 1;
const DWORD DSKMERGE_OUT_NO_PRIMARY_DG  = 1;

const DWORD FTREPLACE_FORCE             = 0x1;
const DWORD FTREPLACE_DELETE_ON_FAIL    = 0x2;

const DWORD CREATE_ASSIGN_ACCESS_PATH   = 0x1;

typedef struct volumespec {
    VOLUMETYPE      type;
    VOLUMELAYOUT    layout;
    REGIONTYPE      partitionType;
    LONGLONG        length;
    LONGLONG        lastKnownState;
}


VOLUME_SPEC;

typedef struct volumeinfo {
    LdmObjectId     id;
    VOLUMETYPE      type;
    VOLUMELAYOUT    layout;
    LONGLONG        length;
    LdmObjectId     fsId;
    unsigned long   memberCount;
    VOLUMESTATUS    status;
    LONGLONG        lastKnownState;
```


Copyright © 2016 Microsoft Corporation

```
    LdmObjectId     taskId;
    unsigned long   vflags;
}
VOLUME_INFO;

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

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;
    [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;
};

typedef struct diskinfo DISK_INFO;

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

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;
```


Copyright © 2016 Microsoft Corporation

```
        unsigned long   rflags;
        unsigned long   currentPartitionNumber;
    };
    typedef struct regioninfo REGION_INFO;

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

    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;
    };
    typedef struct filesysteminfo FILE_SYSTEM_INFO;

    const DWORD ENABLE_VOLUME_COMPRESSION   = 1;

    const DWORD MAX_FS_NAME_SIZE            = 8;
    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;
    };

    typedef struct ifilesysteminfo IFILE_SYSTEM_INFO;

    const unsigned long FSF_FMT_OPTION_COMPRESS     = 0x00000001;
    const unsigned long FSF_FMT_OPTION_LABEL        = 0x00000002;
    const unsigned long FSF_MNT_POINT_SUPPORT       = 0x00000004;
    const unsigned long FSF_REMOVABLE_MEDIA_SUPPORT = 0x00000008;
    const unsigned long FSF_FS_GROW_SUPPORT         = 0x00000010;
    const unsigned long FSF_FS_QUICK_FORMAT_ENABLE  = 0x00000020;
    const unsigned long FSF_FS_ALLOC_SZ_512         = 0x00000040;
    const unsigned long FSF_FS_ALLOC_SZ_1K          = 0x00000080;
    const unsigned long FSF_FS_ALLOC_SZ_2K          = 0x00000100;
    const unsigned long FSF_FS_ALLOC_SZ_4K          = 0x00000200;
    const unsigned long FSF_FS_ALLOC_SZ_8K          = 0x00000400;
    const unsigned long FSF_FS_ALLOC_SZ_16K         = 0x00000800;
    const unsigned long FSF_FS_ALLOC_SZ_32K         = 0x00001000;
    const unsigned long FSF_FS_ALLOC_SZ_64K         = 0x00002000;
    const unsigned long FSF_FS_ALLOC_SZ_128K        = 0x00004000;
    const unsigned long FSF_FS_ALLOC_SZ_256K        = 0x00008000;
    const unsigned long FSF_FS_ALLOC_SZ_OTHER       = 0x00010000;
    const unsigned long FSF_FS_FORMAT_SUPPORTED     = 0x00020000;
    const unsigned long FSF_FS_VALID_BITS           = 0x0003FFFF;

    const long FSTYPE_UNKNOWN               = 0x00000000;
    const long FSTYPE_NTFS                  = 0x00000001;
    const long FSTYPE_FAT                   = 0x00000002;
    const long FSTYPE_FAT32                 = 0x00000003;
    const long FSTYPE_CDFS                  = 0x00000004;
```


Copyright © 2016 Microsoft Corporation

```
const long FSTYPE_UDF                    = 0x00000005;
const long FSTYPE_OTHER                  = 0x80000000;

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

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

typedef struct countedstring COUNTED_STRING;


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

const DWORD ENCAP_INFO_CANT_PROCEED   = 0x1;
const DWORD ENCAP_INFO_NO_FREE_SPACE  = 0x2;
const DWORD ENCAP_INFO_BAD_ACTIVE     = 0x4;

const DWORD ENCAP_INFO_UNKNOWN_PART   = 0x8;
const DWORD ENCAP_INFO_FT_UNHEALTHY   = 0x10;
const DWORD ENCAP_INFO_FT_QUERY_FAILED= 0x20;
const DWORD ENCAP_INFO_FT_HAS_RAID5   = 0x40;
const DWORD ENCAP_INFO_FT_ON_BOOT     = 0x80;

const DWORD ENCAP_INFO_REBOOT_REQD    = 0x100;
const DWORD ENCAP_INFO_CONTAINS_FT    = 0x200;
const DWORD ENCAP_INFO_VOLUME_BUSY    = 0x400;
const DWORD ENCAP_INFO_PART_NR_CHANGE = 0x800;

[ object, uuid(D2D79DF5-3400-11d0-B40B-00AA005FF586),
pointer_default(unique) ]
interface IVolumeClient : IUnknown
{

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

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

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

    HRESULT CreatePartitionAssignAndFormat([in] REGION_SPEC
partitionSpec,
                            [in] wchar_t letter,
```


Copyright © 2016 Microsoft Corporation

```
                            [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 WriteSignature( [in] LdmObjectId diskId,
                            [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,
```


Copyright © 2016 Microsoft Corporation

```
                                [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 Reserved27(
      void
    );

    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,
```


Copyright © 2016 Microsoft Corporation

```
                         [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 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 InitializeDisk([in] LdmObjectId diskId,
                           [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 Reserved_Opnum42( void );

    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,
```


Copyright © 2016 Microsoft Corporation

```
    [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 Reserved_Opnum46( void );

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

        HRESULT Reserved_Opnum48(void);

        HRESULT Reserved_Opnum49(void);

        HRESULT Reserved_Opnum50(void);

        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 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 );

        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 );

        HRESULT QueryChangePartitionNumbers([out] int *oldPartitionNumber,
```


Copyright © 2016 Microsoft Corporation

```
                        [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();
```


Copyright © 2016 Microsoft Corporation

```
    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(D2D79DF7-3400-11d0-B40B-00AA005FF586),
pointer_default(unique) ]
interface IDMNotify : IUnknown
{
    typedef [unique] IDMNotify *LPIDMNOTIFY;

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

}

[ object, uuid(3A410F21-553F-11d1-8E5E-00A0C92C9D5D),
pointer_default(unique) ]
interface IDMRemoteServer : IUnknown
{
     HRESULT CreateRemoteObject([in] unsigned long cMax,
                        [in, max_is(cMax)] wchar_t
*RemoteComputerName);
}

[ object, uuid(4BDAFC52-FE6A-11d2-93F8-00105A11164A),
pointer_default(unique) ]
interface IVolumeClient2 : IUnknown
{
HRESULT GetMaxAdjustedFreeSpace([in] LdmObjectId diskId,
                    [out] LONGLONG* maxAdjustedFreeSpace);
}
```

6.2 Appendix A.2: dmintf3.idl

```
import "ms-dmrp_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 = 1,
PARTITIONSTYLE_GPT = 2
} PARTITIONSTYLE;

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


Copyright © 2016 Microsoft Corporation

```
    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;
```


Copyright © 2016 Microsoft Corporation

```
    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);
```


Copyright © 2016 Microsoft Corporation

```
    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,
```


Copyright © 2016 Microsoft Corporation

```
                    [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,
```


Copyright © 2016 Microsoft Corporation

```
                        [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,
```


Copyright © 2016 Microsoft Corporation

```
                          [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
);
```


Copyright © 2016 Microsoft Corporation

```
    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
);
}
```


Copyright © 2016 Microsoft Corporation

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

- Windows NT 3.1 operating system
- Windows NT 3.5 operating system
- Windows NT 3.51 operating system
- Windows NT 4.0 operating system
- Windows 2000 Server operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Vista operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product 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 (LDM) user interface (UI) and the Diskpart.exe command-line tool.

<2> Section 1.4: The Disk Management Remote Protocol is used by the Windows LDM UI and the Diskpart.exe command-line tool.

<3> Section 1.6: Windows 2000 Server, Windows XP, and Windows Server 2003 implement the Disk Management Remote Protocol. The Windows Server 2003 operating system and Windows Vista implement the VDS Remote Protocol, which is used for disk management for these operating systems. 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 XP, Windows 2000 Server, and Windows Server 2003.
- The IVolumeClient and IVolumeClient2 interfaces are to be used for disk management for Windows 2000 Server. These interfaces can be used for disk management for Windows Server 2003 and Windows XP.
- The IVolumeClient3 interface can be used with Windows XP and Windows Server 2003, but cannot be used with Windows 2000 Server.

Copyright © 2016 Microsoft Corporation

- The IVolumeClient4 interface can be used with Windows Server 2003 but cannot 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, IVolumeClient2, IVolumeClient3, IVolumeClient4	RPC_C_AUTHN_LEVEL_CONNECT RPC_C__IMP_LEVEL_IDENTIFY EOAC_APPID	RPC_C_AUTHN_LEVEL_CONNECT RPC_C__IMP_LEVEL_IMPERSONATE EOAC_NONE	RPC_C_AUTHN_LEVEL_PKT_PRIVACY RPC_C__IMP_LEVEL_IDENTIFY 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 following table explains 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 3.1, Windows NT 3.5, Windows NT 3.51, and Windows NT 4.0 are as follows.

Partition type	Value	Description
PARTITION_EISA	0x12	Extended Industry Standard Architecture (EISA) partition
PARTITION_HIBERNATION	0x84	Hibernation partition for laptops
PARTITION_DIAGNOSTIC	0xA0	Diagnostic partition on some Hewlett-Packard (HP) notebook PCs

Copyright © 2016 Microsoft Corporation

Partition type	Value	Description
PARTITION_DELL	0xDE	Dell partition
PARTITION_IBM	0xFE	IBM initial microprogram load (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

Copyright © 2016 Microsoft Corporation

Opnum	Description
IVolumeClient3 Opnum 2	Default DCOM method Release
IVolumeClient3 Opnum 12	Used only locally by Windows, never remotely
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
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 graphical user interface (GUI) display based on these notifications.

<15> Section 3.2.1.1: In Windows, the **unique identifier (UID)** of the disk object changes when it is converted from basic disk to dynamic disk or from dynamic disk to basic disk.

<16> Section 3.2.4: Windows servers enforce authorization checks. For more information on the authorization requirements for the various methods, see section 2.1.

<17> Section 3.2.4.3: In Windows, all the methods listed that can be implemented asynchronously are implemented as asynchronous methods.

<18> Section 3.2.4.3: For example, in 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. In Windows, the server sends notifications based on 10-percent increments; for example, notifications are sent at 0 percent finished, 10 percent finished, or 20 percent finished.

Copyright © 2016 Microsoft Corporation

<19> Section 3.2.4.4.1.3: A partition cannot be created at the offset zero if the disk is partitioned with either MBR or GPT disk partitioning formats.

<20> Section 3.2.4.4.1.3: MUST be set to zero when sent and MUST be ignored on receipt.

<21> Section 3.2.4.4.1.3: MUST be set to zero when sent and MUST be ignored on receipt.

<22> Section 3.2.4.4.1.3: MUST be set to zero when sent and MUST be ignored on receipt.

<23> Section 3.2.4.4.1.3: A drive letter can be assigned to the partition automatically by the Windows mount point manager depending on several factors, including whether or not NoAutoMount is enabled, whether or not the partition type is recognized by Windows, or whether or not the GPT_BASIC_DATA_ATTRIBUTE_NO_DRIVE_LETTER or GPT_BASIC_DATA_ATTRIBUTE_HIDDEN flags is set.

<24> Section 3.2.4.4.1.3: Windows uses the PARTITION_INFORMATION_EX structure to create and format partitions. For more information about this structure, see [MSDN-PARTITIONINFO].

<25> Section 3.2.4.4.1.4: In Windows, the server does not check the *letterLastKnownState* parameter. Even if the specified drive letter is not present in the list of storage objects, the CreatePartitionAssignAndFormat method creates the partition.

<26> Section 3.2.4.4.1.4: In Windows, if specifies that a drive letter be assigned, the field **tinfo.storageId** is set to 0 even if the partition is created successfully.

<27> Section 3.2.4.4.1.4: MUST be set to zero when sent and MUST be ignored on receipt.

<28> Section 3.2.4.4.1.4: MUST be set to zero when sent and MUST be ignored on receipt.

<29> Section 3.2.4.4.1.4: MUST be set to zero when sent and MUST be ignored on receipt.

<30> Section 3.2.4.4.1.4: The formatting is handled as an asynchronous task.

<31> Section 3.2.4.4.1.6: In Windows, the server does not verify whether *partitionSpec.LastKnownState* matches the **LastKnownState** field of the object.

<32> Section 3.2.4.4.1.6: MUST be set to zero when sent and MUST be ignored on receipt.

<33> Section 3.2.4.4.1.6: MUST be set to zero when sent and MUST be ignored on receipt.

<34> Section 3.2.4.4.1.6: MUST be set to zero when sent and MUST be ignored on receipt.

<35> 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.

<36> Section 3.2.4.4.1.7: MUST be set to zero when sent and MUST be ignored on receipt.

<37> Section 3.2.4.4.1.7: MUST be set to zero when sent and MUST be ignored on receipt.

<38> Section 3.2.4.4.1.7: MUST be set to zero when sent and MUST be ignored on receipt.

<39> Section 3.2.4.4.1.8: MUST be set to zero when sent and MUST be ignored on receipt.

<40> Section 3.2.4.4.1.8: MUST be set to zero when sent and MUST be ignored on receipt.

<41> Section 3.2.4.4.1.8: MUST be set to zero when sent and MUST be ignored on receipt.

<42> Section 3.2.4.4.1.9: MUST be set to zero when sent and MUST be ignored on receipt.

<43> Section 3.2.4.4.1.9: MUST be set to zero when sent and MUST be ignored on receipt.

<44> Section 3.2.4.4.1.9: MUST be set to zero when sent and MUST be ignored on receipt.

Copyright © 2016 Microsoft Corporation

<45> Section 3.2.4.4.1.9: In a Windows implementation, the server always returns the status of the operation as REQ_FAILED.

<46> Section 3.2.4.4.1.10: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<47> Section 3.2.4.4.1.11: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<48> Section 3.2.4.4.1.12: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<49> 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.

<50> Section 3.2.4.4.1.12: MUST be set to zero when sent and MUST be ignored on receipt.

<51> Section 3.2.4.4.1.12: MUST be set to zero when sent and MUST be ignored on receipt.

<52> Section 3.2.4.4.1.12: MUST be set to zero when sent and MUST be ignored on receipt.

<53> Section 3.2.4.4.1.13: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<54> 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.

<55> Section 3.2.4.4.1.13: MUST be set to zero when sent and MUST be ignored on receipt.

<56> Section 3.2.4.4.1.13: MUST be set to zero when sent and MUST be ignored on receipt.

<57> Section 3.2.4.4.1.13: MUST be set to zero when sent and MUST be ignored on receipt.

<58> Section 3.2.4.4.1.13: Note that the new volume that results when breaking a single volume into two separate partitions ~~may~~might automatically get a new drive letter assigned by the operating system.

<59> Section 3.2.4.4.1.14: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<60> Section 3.2.4.4.1.14: MUST be set to zero when sent and MUST be ignored on receipt.

<61> Section 3.2.4.4.1.14: MUST be set to zero when sent and MUST be ignored on receipt.

<62> Section 3.2.4.4.1.14: MUST be set to zero when sent and MUST be ignored on receipt.

<63> Section 3.2.4.4.1.15: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<64> Section 3.2.4.4.1.15: MUST be set to zero when sent and MUST be ignored on receipt.

<65> Section 3.2.4.4.1.15: MUST be set to zero when sent and MUST be ignored on receipt.

<66> Section 3.2.4.4.1.15: MUST be set to zero when sent and MUST be ignored on receipt.

<67> Section 3.2.4.4.1.16: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<68> Section 3.2.4.4.1.16: MUST be set to zero when sent and MUST be ignored on receipt.

<69> Section 3.2.4.4.1.16: MUST be set to zero when sent and MUST be ignored on receipt.

Copyright © 2016 Microsoft Corporation

<70> Section 3.2.4.4.1.16: MUST be set to zero when sent and MUST be ignored on receipt.

<71> Section 3.2.4.4.1.17: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<72> Section 3.2.4.4.1.17: MUST be set to zero when sent and MUST be ignored on receipt.

<73> Section 3.2.4.4.1.17: MUST be set to zero when sent and MUST be ignored on receipt.

<74> Section 3.2.4.4.1.17: MUST be set to zero when sent and MUST be ignored on receipt.

<75> Section 3.2.4.4.1.18: In Windows, the enumeration of drive letter objects excludes the objects with drive letters 'A' and 'B'.

<76> Section 3.2.4.4.1.19: MUST be set to zero when sent and MUST be ignored on receipt.

<77> Section 3.2.4.4.1.19: MUST be set to zero when sent and MUST be ignored on receipt.

<78> Section 3.2.4.4.1.19: MUST be set to zero when sent and MUST be ignored on receipt.

<79> 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.

<80> Section 3.2.4.4.1.20: MUST be set to zero when sent and MUST be ignored on receipt.

<81> Section 3.2.4.4.1.20: MUST be set to zero when sent and MUST be ignored on receipt.

<82> Section 3.2.4.4.1.20: MUST be set to zero when sent and MUST be ignored on receipt.

<83> Section 3.2.4.4.1.21: In Windows, the server returns file system structure without file system information for partitions on a dynamic disk.

<84> Section 3.2.4.4.1.23: The *FILE_SYSTEM_INFO::id* parameter is not used in this case.

<85> 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.

<86> Section 3.2.4.4.1.23: MUST be set to zero when sent and MUST be ignored on receipt.

<87> Section 3.2.4.4.1.23: MUST be set to zero when sent and MUST be ignored on receipt.

<88> Section 3.2.4.4.1.23: MUST be set to zero when sent and MUST be ignored on receipt.

<89> Section 3.2.4.4.1.23: The formatting is handled as an asynchronous task.

<90> Section 3.2.4.4.1.25: This method enumerates the volume extents, not the volume members. A volume member is a **volume plex** for a mirrored volume, or a volume's column\member for a RAID-5 volume.

<91> Section 3.2.4.4.1.25: In Windows, the server returns S_FALSE if the method is successful.

<92> Section 3.2.4.4.1.26: In Windows, if the size of the volume requested is greater than the size of the volume that can be created on the specified disks, the HRESULT returned is S_OK, TASK_INFO::error is set to S_OK, and TASK_INFO::storageId is set to 0 to indicate that the volume was not created.

<93> Section 3.2.4.4.1.26: Note that in Windows the **bNeedContiguous** field in the DISK_SPEC structure is ignored if more than one DISK_SPEC structure is passed in.

<94> Section 3.2.4.4.1.26: MUST be set to zero when sent and MUST be ignored on receipt.

<95> Section 3.2.4.4.1.26: MUST be set to zero when sent and MUST be ignored on receipt.

Copyright © 2016 Microsoft Corporation

<96> Section 3.2.4.4.1.26: MUST be set to zero when sent and MUST be ignored on receipt.

<97> Section 3.2.4.4.1.26: In Windows, the status returned is REQ_STARTED even if the operation has finished successfully.

<98> Section 3.2.4.4.1.26: Windows sends the region-deleted and region-created notification when a region is deleted and created during this operation in MBR disks. Windows does not send the region-deleted and region-created notification when a region is deleted and created during this operation in GPT disks.

<99> Section 3.2.4.4.1.27: In Windows, the server does not check the *letterLastKnownState* parameter. Even if the specified drive letter is not present in the list of storage objects, the CreateVolumeAssignAndFormat method creates the volume.

<100> Section 3.2.4.4.1.27: In a Windows implementation, the field **tinfo.storageId** is set to zero even if the partition is created successfully.

<101> Section 3.2.4.4.1.27: MUST be set to zero when sent and MUST be ignored on receipt.

<102> Section 3.2.4.4.1.27: MUST be set to zero when sent and MUST be ignored on receipt.

<103> Section 3.2.4.4.1.27: MUST be set to zero when sent and MUST be ignored on receipt.

<104> Section 3.2.4.4.1.27: In Windows, the formatting is handled as an asynchronous task.

<105> Section 3.2.4.4.1.30: In Windows, the method returns 0 without growing the volume, if the length specified in **diskList.length** is greater than the available free space on that disk.

<106> Section 3.2.4.4.1.30: MUST be set to zero when sent and MUST be ignored on receipt.

<107> Section 3.2.4.4.1.30: MUST be set to zero when sent and MUST be ignored on receipt.

<108> Section 3.2.4.4.1.30: MUST be set to zero when sent and MUST be ignored on receipt.

<109> 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.

<110> Section 3.2.4.4.1.31: MUST be set to zero when sent and MUST be ignored on receipt.

<111> Section 3.2.4.4.1.31: MUST be set to zero when sent and MUST be ignored on receipt.

<112> Section 3.2.4.4.1.31: MUST be set to zero when sent and MUST be ignored on receipt.

<113> Section 3.2.4.4.1.31: Windows sends the region-deleted and OID-created notification when a region is deleted and created during this operation on MBR disks. Windows does not send the region-deleted and region-created notification when a region is deleted and created during this operation on GPT disks.

<114> 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.

<115> Section 3.2.4.4.1.32: In Windows, the status returned is REQ_STARTED even if the operation has completed successfully.

<116> Section 3.2.4.4.1.32: MUST be set to zero when sent and MUST be ignored on receipt.

<117> Section 3.2.4.4.1.32: MUST be set to zero when sent and MUST be ignored on receipt.

<118> Section 3.2.4.4.1.32: MUST be set to zero when sent and MUST be ignored on receipt.

<119> Section 3.2.4.4.1.32: In Windows, the server sends a multiple-task completion notification if the operation succeeds.

Copyright © 2016 Microsoft Corporation

<120> Section 3.2.4.4.1.33: MUST be set to zero when sent and MUST be ignored on receipt.

<121> Section 3.2.4.4.1.33: MUST be set to zero when sent and MUST be ignored on receipt.

<122> Section 3.2.4.4.1.33: MUST be set to zero when sent and MUST be ignored on receipt.

<123> Section 3.2.4.4.1.34: In Windows, the server does not check the *letterLastKnownState* parameter. Even if the specified drive letter is not present in the list of storage objects, the SplitMirror method splits the volume.

<124> Section 3.2.4.4.1.34: MUST be set to zero when sent and MUST be ignored on receipt.

<125> Section 3.2.4.4.1.34: MUST be set to zero when sent and MUST be ignored on receipt.

<126> Section 3.2.4.4.1.34: MUST be set to zero when sent and MUST be ignored on receipt.

<127> Section 3.2.4.4.1.34: The call fails with LDM_E_VOLUME_IN_USE if the volume cannot be locked.

<128> Section 3.2.4.4.1.35: MUST be set to zero when sent and MUST be ignored on receipt.

<129> Section 3.2.4.4.1.35: MUST be set to zero when sent and MUST be ignored on receipt.

<130> Section 3.2.4.4.1.35: MUST be set to zero when sent and MUST be ignored on receipt.

<131> Section 3.2.4.4.1.35: In Windows implementations, the server does not send the region deletion notification.

<132> Section 3.2.4.4.1.36: In Windows, the server does not verify that the disk is empty when the method is called. Instead, the method sends an asynchronous task notification indicating the task failure if the disk specified is not empty.

<133> Section 3.2.4.4.1.36: MUST be set to zero when sent and MUST be ignored on receipt.

<134> Section 3.2.4.4.1.36: MUST be set to zero when sent and MUST be ignored on receipt.

<135> Section 3.2.4.4.1.36: MUST be set to zero when sent and MUST be ignored on receipt.

<136> Section 3.2.4.4.1.36: The disk conversion is handled as an asynchronous task.

<137> Section 3.2.4.4.1.36: In Windows implementations, the server does not send the region deletion notification.

<138> Section 3.2.4.4.1.37: MUST be set to zero when sent and MUST be ignored on receipt.

<139> Section 3.2.4.4.1.37: MUST be set to zero when sent and MUST be ignored on receipt.

<140> Section 3.2.4.4.1.37: MUST be set to zero when sent and MUST be ignored on receipt.

<141> Section 3.2.4.4.1.37: The disk reactivation operation is handled as an asynchronous task.

<142> Section 3.2.4.4.1.38: MUST be set to zero when sent and MUST be ignored on receipt.

<143> Section 3.2.4.4.1.38: MUST be set to zero when sent and MUST be ignored on receipt.

<144> Section 3.2.4.4.1.38: MUST be set to zero when sent and MUST be ignored on receipt.

<145> Section 3.2.4.4.1.38: The disk import operation is handled as an asynchronous task.

<146> Section 3.2.4.4.1.39: In a Windows implementation, the field status is set to REQ_STARTED even if the operation finished successfully.

Copyright © 2016 Microsoft Corporation

<147> Section 3.2.4.4.1.39: Handling of the DSKMERGE_IN_NO_UNRELATED flag is not implemented in Windows.

<148> Section 3.2.4.4.1.40: In a Windows implementation, the field status is set to REQ_STARTED even if the operation finished successfully.

<149> Section 3.2.4.4.1.40: MUST be set to zero when sent and MUST be ignored on receipt.

<150> Section 3.2.4.4.1.40: MUST be set to zero when sent and MUST be ignored on receipt.

<151> Section 3.2.4.4.1.40: MUST be set to zero when sent and MUST be ignored on receipt.

<152> Section 3.2.4.4.1.41: Windows does not implement this method.

<153> Section 3.2.4.4.1.41: MUST be set to zero when sent and MUST be ignored on receipt.

<154> Section 3.2.4.4.1.41: MUST be set to zero when sent and MUST be ignored on receipt.

<155> Section 3.2.4.4.1.41: MUST be set to zero when sent and MUST be ignored on receipt.

<156> Section 3.2.4.4.1.42: MUST be set to zero when sent and MUST be ignored on receipt.

<157> Section 3.2.4.4.1.42: MUST be set to zero when sent and MUST be ignored on receipt.

<158> Section 3.2.4.4.1.42: MUST be set to zero when sent and MUST be ignored on receipt.

<159> Section 3.2.4.4.1.43: MUST be set to zero when sent and MUST be ignored on receipt.

<160> Section 3.2.4.4.1.43: MUST be set to zero when sent and MUST be ignored on receipt.

<161> Section 3.2.4.4.1.43: MUST be set to zero when sent and MUST be ignored on receipt.

<162> Section 3.2.4.4.1.43: In Windows, the status returned is REQ_STARTED even if the operation has been finished successfully.

<163> Section 3.2.4.4.1.44: A notification that a task has been modified is sent for failure cases. A task-modified notification is not usually sent when a task fails, but parameter-validation failure is an exception.

<164> Section 3.2.4.4.1.44: MUST be set to zero when sent and MUST be ignored on receipt.

<165> Section 3.2.4.4.1.44: MUST be set to zero when sent and MUST be ignored on receipt.

<166> Section 3.2.4.4.1.44: MUST be set to zero when sent and MUST be ignored on receipt.

<167> Section 3.2.4.4.1.45: MUST be set to zero when sent and MUST be ignored on receipt.

<168> Section 3.2.4.4.1.45: MUST be set to zero when sent and MUST be ignored on receipt.

<169> Section 3.2.4.4.1.45: This **tinfo::Status** field is returned as REQ_STARTED rather than REQ_COMPLETED.

<170> Section 3.2.4.4.1.45: In a Windows implementation, the server does not send task completion notification.

<171> Section 3.2.4.4.1.45: In a Windows implementation, the server does not send notifications for deletion of region objects of the old basic disks.

<172> Section 3.2.4.4.1.45: 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

Copyright © 2016 Microsoft Corporation

"OldPartitionNumber" and "NewPartitionNumber." After the server has updated the boot settings as necessary, it deletes the registry entries for "OldPartitionNumber" and "NewPartitionNumber".

<173> Section 3.2.4.4.1.47: Windows stores the boot partition change information in the registry under "HKLM\SYSTEM\CurrentControlSet\Services\dmio\Partition Info" as DWORD values "OldPartitionNumber" and "NewPartitionNumber."

<174> Section 3.2.4.4.1.48: 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.

<175> Section 3.2.4.4.1.49: 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.

<176> Section 3.2.4.4.1.50: In Windows, this method is not implemented and returns E_FAIL.

<177> Section 3.2.4.4.1.53: Windows always returns S_FALSE.

<178> Section 3.2.4.4.1.53: No flags are defined or returned.

<179> Section 3.2.4.4.1.53: Windows does not return this information from the server. Windows clients on Windows 2000 operating system and Windows XP 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.

<180> Section 3.2.4.4.1.54: In Windows, the LDM UI client checks the value of *ulIDLVersion* to be equal to the version of the IDL file with which the client was built and will disconnect from the server if the *ulIDLVersion* is not the same.

<181> Section 3.2.4.4.1.54: This flag is never set by the 32-bit version of Windows 2000 Server, Windows Server 2003, or Windows XP. This flag is set by the 64-bit version of Windows XP and Windows Server 2003.

<182> Section 3.2.4.4.1.54: This flag is never set by Windows 2000 Server, Windows Server 2003, or Windows XP.

<183> Section 3.2.4.4.1.54: This flag is set only by Windows 2000 Server.

<184> Section 3.2.4.4.1.54: In Windows, the server responds to all client messages even if the Initialize method has not been called by the client, with the limitation that the client cannot receive any notifications from the server until the Initialize method has been called.

<185> Section 3.2.4.4.1.54: In Windows XP operating system Service Pack 2 (SP2) and Windows Server 2003 operating system with Service Pack 1 (SP1), if *cRemote* parameter is nonzero, the server uses server machine account authentication to make calls to the IDMNotify interface that is specified by *notificationInterface*.

<186> Section 3.2.4.4.1.55: In Windows, the server responds to all client messages — even after the Uninitialize method has been called by the client. However, after the Uninitialize method has been called by the client, the client cannot receive any further notifications from the server.

<187> Section 3.2.4.4.1.59: There is no operating system support for this method, so it is not used by Windows.

<188> Section 3.2.4.4.2.1: In Windows, if the *diskId* is not in the list of storage objects, the server causes the method to succeed without setting the *maxAdjustedFreeSpace* parameter.

<189> Section 3.2.4.4.3: IVolumeClient3 methods are not implemented in Windows 2000 Server.

Copyright © 2016 Microsoft Corporation

<190> Section 3.2.4.4.3.3: Windows uses the PARTITION_INFORMATION_EX structure to create and format partitions. For more information about this structure, see [MSDN-PARTITIONINFO].

<191> Section 3.2.4.4.3.7: MUST be set to zero when sent and MUST be ignored on receipt.

<192> Section 3.2.4.4.3.7: MUST be set to zero when sent and MUST be ignored on receipt.

<193> Section 3.2.4.4.3.7: MUST be set to zero when sent and MUST be ignored on receipt.

<194> Section 3.2.4.4.3.10: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<195> Section 3.2.4.4.3.11: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<196> Section 3.2.4.4.3.12: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<197> Section 3.2.4.4.3.13: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<198> Section 3.2.4.4.3.14: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<199> Section 3.2.4.4.3.15: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<200> Section 3.2.4.4.3.16: In Windows, FT volumes on basic disks can only be created in Windows NT 4.0.

<201> Section 3.2.4.4.3.20: In Windows, the server returns file system structure without file system information for partitions on a dynamic disks.

<202> Section 3.2.4.4.3.32: MUST be set to zero when sent and MUST be ignored on receipt.

<203> Section 3.2.4.4.3.32: MUST be set to zero when sent and MUST be ignored on receipt.

<204> Section 3.2.4.4.3.32: MUST be set to zero when sent and MUST be ignored on receipt.

<205> Section 3.2.4.4.3.32: In a Windows implementation, the server does not send the task completion notification.

<206> Section 3.2.4.4.3.32: In Windows, the lastKnownState of the disk object does not change, even though the disk object is modified.

<207> Section 3.2.4.4.3.33: MUST be set to zero when sent and MUST be ignored on receipt.

<208> Section 3.2.4.4.3.33: MUST be set to zero when sent and MUST be ignored on receipt.

<209> Section 3.2.4.4.3.33: MUST be set to zero when sent and MUST be ignored on receipt.

<210> Section 3.2.4.4.3.33: In a Windows implementation, the server does not send the task completion notification.

<211> Section 3.2.4.4.3.38: MUST be set to zero when sent and MUST be ignored on receipt.

<212> Section 3.2.4.4.3.38: MUST be set to zero when sent and MUST be ignored on receipt.

<213> Section 3.2.4.4.3.38: MUST be set to zero when sent and MUST be ignored on receipt.

<214> Section 3.2.4.4.3.38: In Windows implementations, the server does not send the region deletion notification.

Copyright © 2016 Microsoft Corporation

<215> Section 3.2.4.4.3.44: Windows does not implement this method.

<216> Section 3.2.4.4.3.48: MUST be set to zero when sent and MUST be ignored on receipt.

<217> Section 3.2.4.4.3.48: MUST be set to zero when sent and MUST be ignored on receipt.

<218> Section 3.2.4.4.3.48: MUST be set to zero when sent and MUST be ignored on receipt.

<219> Section 3.2.4.4.3.48: In a Windows implementation, the server does not send task completion notification.

<220> Section 3.2.4.4.3.48: In a Windows implementation, the server does not send notifications for deletion of region objects of the old basic disks.

<221> 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).

<222> Section 3.2.4.4.3.52: The GetDontShow method retrieves a 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 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.

<223> Section 3.2.4.4.3.56: No flags are defined, and this parameter is always initialized to 0.

<224> Section 3.2.4.4.3.57: This flag is never set by the 32-bit version of Windows 2000 Server, Windows Server 2003, or Windows XP. This flag is set by the 64-bit version of Windows XP and Windows Server 2003.

<225> Section 3.2.4.4.3.57: This flag is never set by Windows 2000 Server, Windows Server 2003, or Windows XP.

<226> Section 3.2.4.4.3.57: This flag is set only by Windows 2000 Server.

<227> Section 3.2.4.4.4: IVolumeClient4 methods are not implemented in Windows 2000 Server or Windows XP.

<228> Section 3.2.4.4.4.1: The Windows volume manager keeps track of the dynamic disks present on a system and displays disks that are no longer present as missing.

<229> Section 3.2.6: Windows servers do register for such notifications.

<230> Section 5: For Windows-specific default security configuration please see [MSDN-DefAccPerms] and [MSDN-AccPerms].

Copyright © 2016 Microsoft Corporation

8 Appendix C: IDMNotify::ObjectsChanged

```
void CClientClass::ObjectsChanged( DWORD dwByteCount, BYTE *pByte)
{
    DWORD   dwNotifSize     = 0;
    DWORD   dwCopySize      = 0;
    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;
```


Copyright © 2016 Microsoft Corporation

```
    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
```


Copyright © 2016 Microsoft Corporation

```
    * 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;
```


Copyright © 2016 Microsoft Corporation

```
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;
```


Copyright © 2016 Microsoft Corporation

```
        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, 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;
```


Copyright © 2016 Microsoft Corporation

```
            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

}
```


Copyright © 2016 Microsoft Corporation

9 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

Copyright © 2016 Microsoft Corporation

10 Index

A

AbortTask method (section 3.2.4.4.1.52 153, section 3.2.4.4.3.55 205)
Abstract data model
  client 60
  server 76
AddAccessPath method (section 3.2.4.4.1.63 161, section 3.2.4.4.3.66 209)
AddMirror method (section 3.2.4.4.1.32 124, section 3.2.4.4.3.35 189)
Applicability 17
AssignDriveLetter method (section 3.2.4.4.1.19 107, section 3.2.4.4.3.19 177)

C

Capability negotiation 17
Change tracking 265
Client
  abstract data model 60
  higher-layer triggered events 65
  local events 76
  message processing 60
  sequencing rules 60
  timer events 76
  timers 60
Common data types 18
Connection loss 73
Connections (section 3.2.1.2 77, section 3.2.3.2 78)
COUNTED_STRING structure 43
CreatePartition method (section 3.2.4.4.1.3 86, section 3.2.4.4.3.3 168)
CreatePartitionAssignAndFormat method (section 3.2.4.4.1.4 87, section 3.2.4.4.3.4 168)
CreatePartitionAssignAndFormatEx method (section 3.2.4.4.1.5 89, section 3.2.4.4.3.5 169)
CreatePartitionsForVolume method 185
CreateRemoteObject method 212
CreateVolume method (section 3.2.4.4.1.26 115, section 3.2.4.4.3.26 181)
CreateVolumeAssignAndFormat method (section 3.2.4.4.1.27 117, section 3.2.4.4.3.27 182)
CreateVolumeAssignAndFormatEx method (section 3.2.4.4.1.28 119, section 3.2.4.4.3.28 182)
Creating a partition example 217
Creating a volume example 221

D

Data model - abstract
  client 60
  server 76
Data types
  common 18
  common - overview 18
  IDMNotify interface 58
  IDMRemoteServer interface 58
  IVolumeClient interface 45
  IVolumeClient2 interface 50
  IVolumeClient3 interface 50
  IVolumeClient4 interface 58
DeleteAccessPath method (section 3.2.4.4.1.64 161, section 3.2.4.4.3.67 210)
DeletePartition method (section 3.2.4.4.1.6 90, section 3.2.4.4.3.6 170)
DeletePartitionNumberInfoFromRegistry method (section 3.2.4.4.1.47 150, section 3.2.4.4.3.50 203)
DeletePartitionsForVolume method 187
DeleteVolume method (section 3.2.4.4.1.31 123, section 3.2.4.4.3.31 185)
Deleting a partition example 219
Deleting a volume example 223
Disk arrival 213
Disk layout change 213
Disk removal 213

Copyright © 2016 Microsoft Corporation

DISK_INFO structure 45
DISK_INFO_EX structure 51
DISK_SPEC structure 37
DiskMerge method (section 3.2.4.4.1.40 136, section 3.2.4.4.3.43 194)
DiskMergeQuery method (section 3.2.4.4.1.39 135, section 3.2.4.4.3.42 193)
Disks 67
dmintf.idl 227
dmintf3.idl 238
DMNotify::ObjectsChanged 259
DMNOTIFY_INFO_TYPE enumeration 58
dmProgressType enumeration 43
Drive letter arrival 213
Drive letter removal 214
Drive letters 66
DRIVE_LETTER_INFO structure 38

E

Eject method (section 3.2.4.4.1.9 94, section 3.2.4.4.3.9 172)
EncapsulateDisk method 146
EncapsulateDiskEx method 199
EnumAccessPath method (section 3.2.4.4.1.61 159, section 3.2.4.4.3.64 208)
EnumAccessPathForVolume method (section 3.2.4.4.1.62 160, section 3.2.4.4.3.65 209)
EnumDiskRegions method 85
EnumDiskRegionsEx method 167
EnumDisks method 84
EnumDisksEx method 166
EnumDriveLetters method (section 3.2.4.4.1.18 106, section 3.2.4.4.3.18 177)
EnumLocalFileSystems method (section 3.2.4.4.1.21 110, section 3.2.4.4.3.21 179)
EnumTasks method (section 3.2.4.4.1.50 152, section 3.2.4.4.3.53 204)
EnumVolumeMembers method (section 3.2.4.4.1.25 114, section 3.2.4.4.3.25 181)
EnumVolumes method (section 3.2.4.4.1.24 114, section 3.2.4.4.3.24 180)
Examples 215
  creating a partition 217
  creating a volume 221
  deleting a partition 219
  deleting a volume 223
  starting a new session on a local or remote server 215
  starting a new session on a remote server using the idmremoteserver interface 216

F

Fields - vendor-extensible 17
File system change 213
File systems 67
FILE_SYSTEM_INFO structure 39
Format method (section 3.2.4.4.1.23 112, section 3.2.4.4.3.23 180)
FreeDriveLetter method (section 3.2.4.4.1.20 109, section 3.2.4.4.3.20 178)
FTBreakMirror method (section 3.2.4.4.1.13 98, section 3.2.4.4.3.13 174)
FTDeleteVolume method (section 3.2.4.4.1.12 97, section 3.2.4.4.3.12 173)
FTEnumLogicalDiskMembers method (section 3.2.4.4.1.11 96, section 3.2.4.4.3.11 173)
FTEnumVolumes method (section 3.2.4.4.1.10 95, section 3.2.4.4.3.10 172)
FTRegenerateParityStripe method (section 3.2.4.4.1.15 101, section 3.2.4.4.3.15 175)
FTReplaceMirrorPartition method (section 3.2.4.4.1.16 102, section 3.2.4.4.3.16 175)
FTReplaceParityStripePartition method (section 3.2.4.4.1.17 104, section 3.2.4.4.3.17 176)
FTResyncMirror method (section 3.2.4.4.1.14 100, section 3.2.4.4.3.14 174)
Full IDL (section 6 227, section 6.1 227, section 6.2 238)

G

GetDontShow method (section 3.2.4.4.1.49 151, section 3.2.4.4.3.52 203)
GetEncapsulateDiskInfo method 142
GetEncapsulateDiskInfoEx method 196
GetInstalledFileSystems method (section 3.2.4.4.1.22 111, section 3.2.4.4.3.22 179)
GetMaxAdjustedFreeSpace method (section 3.2.4.4.2.1 163, section 3.2.4.4.3.34 188)

Copyright © 2016 Microsoft Corporation

GetTaskDetail method (section 3.2.4.4.1.51 152, section 3.2.4.4.3.54 204)
GetVolumeDeviceName method 211
GetVolumeMountName method (section 3.2.4.4.1.29 120, section 3.2.4.4.3.29 184)
Glossary 9
GrowVolume method (section 3.2.4.4.1.30 121, section 3.2.4.4.3.30 184)

H

Higher-layer triggered events
  client 65
  server 79
HrGetErrorData method (section 3.2.4.4.1.53 154, section 3.2.4.4.3.56 205)

I

IDL (section 6 227, section 6.1 227, section 6.2 238)
IDMNotify interface
  data types 58
  overview 58
IDMNotify methods 74
IDMRemoteServer interface
  data types 58
  overview 58
IFILE_SYSTEM_INFO structure 40
Implementer - security considerations 226
ImportDiskGroup method (section 3.2.4.4.1.38 133, section 3.2.4.4.3.41 192)
Informative references 15
Initialization
  client 60
  server 78
Initialize method (section 3.2.4.4.1.54 155, section 3.2.4.4.3.57 206)
InitializeDisk method 129
InitializeDiskEx method 190
InitializeDiskStyle method 170
Introduction 9
IVolumeClient interface 45
IVolumeClient2 interface 50
IVolumeClient3 interface (section 2.5 50, section 2.5.1 50)
IVolumeClient4 interface
  data types 58
  overview 58

L

LDMACTION enumeration 59
List of client connections (section 3.2.1.2 77, section 3.2.3.2 78)
List of current tasks (section 3.2.1.3 78, section 3.2.3.3 78)
List of storage objects (section 3.2.1.1 76, section 3.2.3.1 78)
Local events
  client 76
  server 212
Loss of connection 73

M

MarkActivePartition method (section 3.2.4.4.1.8 93, section 3.2.4.4.3.8 171)
MAX_FS_NAME_SIZE 31
Media arrival 214
Media removal 214
MERGE_OBJECT_INFO structure 44
Message processing
  client 60
  server 78
Messages
  common data types 18

Copyright © 2016 Microsoft Corporation

data types 18
details (section 3.1.4.4 74, section 3.2.4.4 81)
overview 18
transport 18
Methods with prerequisites 65

N

Normative references 15

O

ObjectsChanged method 74
Overview (synopsis) 16

P

Parameters to IVolumeClient and IVolumeClient3 65
PARTITION_OS2_BOOT 45
Partitions 70
PARTITIONSTYLE enumeration 50
Preconditions 17
Prerequisites 17
Processing notifications - server to client 73
Processing server replies to method calls 73
Product behavior 246
Protocol Details
overview 60

Q

QueryChangePartitionNumbers method (section 3.2.4.4.1.46 149, section 3.2.4.4.3.49 202)

R

ReAttachDisk method (section 3.2.4.4.1.41 138, section 3.2.4.4.3.44 194)
ReConnectDisk method (section 3.2.4.4.1.37 132, section 3.2.4.4.3.40 192)
References 15
informative 15
normative 15
Refresh method (section 3.2.4.4.1.56 157, section 3.2.4.4.3.59 207)
RefreshEx method 210
RefreshFileSys method (section 3.2.4.4.1.58 158, section 3.2.4.4.3.61 208)
REGION_INFO structure 48
REGION_INFO_EX structure 55
REGION_SPEC structure 37
REGIONSTATUS enumeration 34
REGIONTYPE enumeration 32
Relationship to other protocols 17
Relationships between storage objects 66
RemoveMirror method (section 3.2.4.4.1.33 126, section 3.2.4.4.3.36 189)
ReplaceRaid5Column method (section 3.2.4.4.1.42 139, section 3.2.4.4.3.45 195)
REQSTATUS enumeration 33
RescanDisks method (section 3.2.4.4.1.57 158, section 3.2.4.4.3.60 207)
RestartVolume method (section 3.2.4.4.1.43 141, section 3.2.4.4.3.46 195)
Rules - asynchronous tasks 79
Rules - modify storage objects list 79
Rules - synchronous tasks 79

S

SecureSystemPartition method (section 3.2.4.4.1.59 158, section 3.2.4.4.3.62 208)
Security 226
Security - implementer considerations 226

Copyright © 2016 Microsoft Corporation

Sequencing rules
client 60
server 78
Server
abstract data model 76
higher-layer triggered events 79
initialization 78
local events 212
message processing 78
sequencing rules 78
timer events 212
timers 78
SetDontShow method (section 3.2.4.4.1.48 150, section 3.2.4.4.3.51 203)
ShutDownSystem method (section 3.2.4.4.1.60 159, section 3.2.4.4.3.63 208)
SplitMirror method (section 3.2.4.4.1.34 128, section 3.2.4.4.3.37 190)
Standards assignments 17
Starting a new session on a local or remote server example 215
Starting a new session on a remote server using the idmremoteserver interface example 216
Storage objects (section 3.2.1.1 76, section 3.2.3.1 78)

T

TASK_INFO structure 42
Tasks 73
Tasks currently executed (section 3.2.1.3 78, section 3.2.3.3 78)
Timer events
client 76
server 212
Timers
client 60
server 78
Tracking changes 265
Transport 18
Triggered events - higher-layer
client 65
server 79

U

Uninitialize method (section 3.2.4.4.1.55 157, section 3.2.4.4.3.58 207)
UninitializeDisk method (section 3.2.4.4.1.36 131, section 3.2.4.4.3.39 192)

V

Vendor-extensible fields 17
Versioning 17
VOLUME_INFO structure 35
VOLUME_SPEC structure 35
VOLUMELAYOUT enumeration 33
Volumes 70
VOLUMESTATUS enumeration 34
VOLUMETYPE enumeration 32

W

WriteSignature method 92

Copyright © 2016 Microsoft Corporation