[MS-VDS]:
Virtual Disk Service (VDS) Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
12/18/2006	0.1		MCPP Milestone 2 Initial Availability
03/02/2007	1.0		MCPP Milestone 2
04/03/2007	1.1		Monthly release
05/11/2007	1.2		Monthly release
06/01/2007	2.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
07/03/2007	3.0	Major	Added IVdsVolume::SetFlags and IVdsVolume::ClearFlags.
07/20/2007	3.0.1	Editorial	Revised and edited the technical content.
08/10/2007	3.0.2	Editorial	Revised and edited the technical content.
09/28/2007	4.0	Major	Added two interfaces.
10/23/2007	4.0.1	Editorial	Revised and edited the technical content.
11/30/2007	4.0.2	Editorial	Revised and edited the technical content.
01/25/2008	4.0.3	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction 11
1.1 Glossary 11
1.2 References 15
1.2.1 Normative References 15
1.2.2 Informative References 16
1.3 Protocol Overview (Synopsis) 16
1.3.1 Method Invocation 17
1.3.1.1 Hierarchy Overview 17
1.3.1.2 Method Sequencing Requirements 17
1.3.1.3 Storage Object Relationships 17
1.3.2 Service and Providers 20
1.3.3 Packs 21
1.3.4 Disks 21
1.3.5 Volumes 23
1.3.6 File Systems, Drive Letters, and Access Paths 23
1.4 Relationship to Other Protocols 24
1.5 Prerequisites and Preconditions 24
1.6 Applicability Statement 24
1.7 Versioning and Capability Negotiation 24
1.8 Vendor-Extensible Fields 24
1.9 Standards Assignments 24

2 Messages 27
2.1 Transport 27
2.2 Message Syntax 27
2.2.1 Common Data Types 27
2.2.1.1 Data Types 27
2.2.1.1.1 ULONGLONG 27
2.2.1.1.2 DWORD 27
2.2.1.1.3 VDS_OBJECT_ID 27
2.2.1.1.4 VDS_LUN_INFORMATION 28
2.2.1.2 Enumerations 28
2.2.1.2.1 VDS_HEALTH 28
2.2.1.2.2 VDS_NOTIFICATION_TARGET_TYPE 29
2.2.1.2.3 VDS_ASYNC_OUTPUT_TYPE 29
2.2.1.2.4 VDS_STORAGE_BUS_TYPE 30
2.2.1.2.5 VDS_STORAGE_IDENTIFIER_CODE_SET 31
2.2.1.2.6 VDS_STORAGE_IDENTIFIER_TYPE 31
2.2.1.2.7 VDS_INTERCONNECT_ADDRESS_TYPE 32
2.2.1.2.8 VDS_FILE_SYSTEM_TYPE 33
2.2.1.2.9 VDS_FILE_SYSTEM_FLAG 33
2.2.1.2.10 VDS_FILE_SYSTEM_PROP_FLAG 35
2.2.1.2.11 VDS_FILE_SYSTEM_FORMAT_SUPPORT_FLAG 35
2.2.1.2.12 VDS_DISK_EXTENT_TYPE 35
2.2.1.2.13 VDS_PARTITION_STYLE 36
2.2.1.2.14 VDS_PARTITION_FLAG 36
2.2.1.2.15 VDS_VOLUME_TYPE 36
2.2.1.3 Structures 37
2.2.1.3.1 VDS_PACK_NOTIFICATION 37
2.2.1.3.2 VDS_DISK_NOTIFICATION 37
2.2.1.3.3 VDS_TRANSITION_STATE 38
2.2.1.3.4 VDS_VOLUME_NOTIFICATION 38

2.2.1.3.5 VDS_PARTITION_NOTIFICATION 39
2.2.1.3.6 VDS_DRIVE_LETTER_NOTIFICATION 40
2.2.1.3.7 VDS_FILE_SYSTEM_NOTIFICATION 40
2.2.1.3.8 VDS_MOUNT_POINT_NOTIFICATION 41
2.2.1.3.9 VDS_NOTIFICATION 41
2.2.1.3.10 VDS_ASYNC_OUTPUT 42
2.2.1.3.11 VDS_PARTITION_INFO_MBR 43
2.2.1.3.12 VDS_PARTITION_INFO_GPT 43
2.2.1.3.13 VDS_STORAGE_IDENTIFIER 44
2.2.1.3.14 VDS_STORAGE_DEVICE_ID_DESCRIPTOR 45
2.2.1.3.15 VDS_INTERCONNECT 45
2.2.1.3.16 VDS_LUN_INFORMATION 45
2.2.1.3.17 VDS_FILE_SYSTEM_PROP 46
2.2.1.3.18 VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP 47
2.2.1.3.19 VDS_DISK_EXTENT 48
2.2.1.3.20 VDS_PARTITION_PROP 48
2.2.1.3.21 VDS_INPUT_DISK 49
2.2.1.3.22 CREATE_PARTITION_PARAMETERS 49
2.2.2 Interface-Specific Data Types 51
2.2.2.1 IVdsService Data Types 51
2.2.2.1.1 Data Types 51
2.2.2.1.1.1 MAX_FS_NAME_SIZE 51
2.2.2.1.2 Enumerations 51
2.2.2.1.2.1 VDS_OBJECT_TYPE 51
2.2.2.1.2.2 VDS_SERVICE_FLAG 52
2.2.2.1.2.3 VDS_QUERY_PROVIDER_FLAG 52
2.2.2.1.2.4 VDS_DRIVE_LETTER_FLAG 53
2.2.2.1.3 Structures 53
2.2.2.1.3.1 VDS_SERVICE_PROP 53
2.2.2.1.3.2 VDS_DRIVE_LETTER_PROP 53
2.2.2.1.3.3 VDS_FILE_SYSTEM_TYPE_PROP 54
2.2.2.2 IVdsServiceIscsi Data Types 54
2.2.2.2.1 Structures 54
2.2.2.2.1.1 VDS_ISCSI_SHARED_SECRET 54
2.2.2.3 IVdsHbaPort Data Types 55
2.2.2.3.1 Enumerations 55
2.2.2.3.1.1 VDS_HBAPORT_TYPE 55
2.2.2.3.1.2 VDS_HBAPORT_STATUS 56
2.2.2.3.1.3 VDS_HBAPORT_SPEED_FLAG 57
2.2.2.3.1.4 VDS_PATH_STATUS 57
2.2.2.3.2 Structures 58
2.2.2.3.2.1 VDS_WWN 58
2.2.2.3.2.2 VDS_HBAPORT_PROP 58
2.2.2.4 IVdsIscsiInitiatorAdapter Data Types 59
2.2.2.4.1 Structures 59
2.2.2.4.1.1 VDS_ISCSI_INITIATOR_ADAPTER_PROP 59
2.2.2.5 IVdsIscsiInitiatorPortal Data Types 59
2.2.2.5.1 Enumerations 59
2.2.2.5.1.1 VDS_IPADDRESS_TYPE 59
2.2.2.5.2 Structures 60
2.2.2.5.2.1 VDS_IPADDRESS 60
2.2.2.5.2.2 VDS_ISCSI_INITIATOR_PORTAL_PROP 60
2.2.2.6 IVdsProvider Data Types 61
2.2.2.6.1 Enumerations 61
2.2.2.6.1.1 VDS_PROVIDER_TYPE 61

2.2.2.6.1.2 VDS_PROVIDER_FLAG 61
2.2.2.6.2 Structures 62
2.2.2.6.2.1 VDS_PROVIDER_PROP 62
2.2.2.7 IVdsPack Data Types 63
2.2.2.7.1 Enumerations 63
2.2.2.7.1.1 VDS_PACK_STATUS 63
2.2.2.7.1.2 VDS_PACK_FLAG 63
2.2.2.7.2 Structures 64
2.2.2.7.2.1 VDS_PACK_PROP 64
2.2.2.8 IVdsDisk Data Types 64
2.2.2.8.1 Enumerations 64
2.2.2.8.1.1 VDS_DISK_STATUS 64
2.2.2.8.1.2 VDS_DISK_FLAG 65
2.2.2.8.1.3 VDS_LUN_RESERVE_MODE 65
2.2.2.8.2 Structures 66
2.2.2.8.2.1 VDS_DISK_PROP 66
2.2.2.9 IVdsAdvancedDisk Data Types 68
2.2.2.9.1 Structures 68
2.2.2.9.1.1 CHANGE_ATTRIBUTES_PARAMETERS 68
2.2.2.10 IVdsAdvancedDisk2 Data Types 69
2.2.2.10.1 Structures 69
2.2.2.10.1.1 CHANGE_PARTITION_TYPE_PARAMETERS 69
2.2.2.11 IVdsVolume Data Types 70
2.2.2.11.1 Enumerations 70
2.2.2.11.1.1 VDS_VOLUME_STATUS 70
2.2.2.11.1.2 VDS_VOLUME_FLAG 70
2.2.2.11.2 Structures 72
2.2.2.11.2.1 VDS_VOLUME_PROP 72
2.2.2.12 IVdsVolumeMF Data Types 73
2.2.2.12.1 Data Types 73
2.2.2.12.1.1 MAX_PATH 73
2.2.2.12.2 Structures 73
2.2.2.12.2.1 VDS_REPARSE_POINT_PROP 73
2.2.2.13 IVdsVolumePlex Data Types 74
2.2.2.13.1 Enumeration 74
2.2.2.13.1.1 VDS_VOLUME_PLEX_TYPE 74
2.2.2.13.1.2 VDS_VOLUME_PLEX_STATUS 74
2.2.2.13.2 Structures 75
2.2.2.13.2.1 VDS_VOLUME_PLEX_PROP 75

3 Protocol Details 76
3.1 Interfaces 76
3.1.1 Enumeration Object Interfaces 76
3.1.1.1 IEnumVdsObject Interface 76
3.1.2 Callback Object Interfaces 77
3.1.2.1 IVdsAdviseSink Interface 77
3.1.3 Asynchronous Operation Object Interfaces 77
3.1.3.1 IVdsAsync Interface 77
3.1.4 Service Loader Interfaces 78
3.1.4.1 IVdsServiceLoader Interface 78
3.1.5 Service Object Interfaces 78
3.1.5.1 IVdsService Interface 78
3.1.5.2 IVdsServiceInitialization Interface 80
3.1.5.3 IVdsServiceUninstallDisk Interface 80
3.1.5.4 IVdsServiceHba Interface 81

3.1.5.5 IVdsServiceIscsi Interface 81
3.1.6 HBA Port Object Interfaces 82
3.1.6.1 IVdsHbaPort Interface 82
3.1.7 Initiator Adapter Object Interfaces 82
3.1.7.1 IVdsIscsiInitiatorAdapter Interface 82
3.1.8 Initiator Portal Object Interfaces 83
3.1.8.1 IVdsIscsiInitiatorPortal Interface 83
3.1.9 Provider Object Interfaces 84
3.1.9.1 IVdsProvider Interface 84
3.1.9.2 IVdsSwProvider Interface 84
3.1.9.3 IVdsHwProvider Interface 84
3.1.10 SubSystem Object Interfaces 85
3.1.10.1 IVdsSubSystemImportTarget Interface 85
3.1.11 Pack Object Interfaces 85
3.1.11.1 IVdsPack Interface 85
3.1.11.2 IVdsPack2 Interface 86
3.1.12 Disk Object Interfaces 86
3.1.12.1 IVdsDisk Interface 87
3.1.12.2 IVdsDisk2 Interface 87
3.1.12.3 IVdsAdvancedDisk Interface 88
3.1.12.4 IVdsAdvancedDisk2 Interface 88
3.1.12.5 IVdsCreatePartitionEx Interface 89
3.1.12.6 IVdsDiskPartitionMF Interface 89
3.1.12.7 IVdsRemovable Interface 90
3.1.13 Volume Object Interfaces 90
3.1.13.1 IVdsVolume Interface 90
3.1.13.2 IVdsVolumeMF Interface 91
3.1.13.3 IVdsVolumeMF2 Interface 92
3.1.13.4 IVdsVolumeShrink Interface 92
3.1.13.5 IVdsVolumeOnline Interface 93
3.1.14 Volume Plex Object Interfaces 93
3.1.14.1 IVdsVolumePlex Interface 93
3.2 Client Details 94
3.2.1 Abstract Data Model 94
3.2.1.1 Notification Callback Objects 94
3.2.2 Timers 94
3.2.3 Initialization 94
3.2.4 Message Processing Events and Sequencing Rules 95
3.2.4.1 Processing Server Replies to Method Calls 95
3.2.4.2 Processing Notifications Sent from the Server to the Client 95
3.2.4.3 IVdsAdviseSink Methods 95
3.2.4.3.1 IVdsAdviseSink::OnNotify (Opnum 3) 95
3.2.5 Timer Events 95
3.2.6 Other Local Events 95
3.3 Server Details 96
3.3.1 Abstract Data Model 96
3.3.1.1 Service Object 96
3.3.1.2 Storage Management Objects 96
3.3.1.3 Enumeration of Objects 99
3.3.1.4 Notification Callback Objects 100
3.3.1.5 Asynchronous Tasks 101
3.3.2 Timers 102
3.3.3 Initialization 102
3.3.3.1 Storage Management Objects 102
3.3.3.2 Notification Callback Objects 102

3.3.4 Higher-Layer Triggered Events 102
3.3.5 Message Processing Events and Sequencing Rules 103
3.3.5.1 Sequencing Rules 103
3.3.5.1.1 Adding Pack Objects for Dynamic Providers 103
3.3.5.1.2 Removing Pack Objects for Dynamic Providers 103
3.3.5.1.3 Adding Disk Objects 103
3.3.5.1.4 Removing Disk Objects 105
3.3.5.1.5 Adding Volume Objects 106
3.3.5.1.6 Removing Volume Objects 106
3.3.5.1.7 Handling Asynchronous Tasks 107
3.3.5.2 Message Processing Details 108
3.3.5.2.1 IEnumVdsObject Methods 108
3.3.5.2.1.1 IEnumVdsObject::Next (Opnum 3) 108
3.3.5.2.1.2 IEnumVdsObject::Skip (Opnum 4) 109
3.3.5.2.1.3 IEnumVdsObject::Reset (Opnum 5) 110
3.3.5.2.1.4 IEnumVdsObject::Clone (Opnum 6) 110
3.3.5.2.2 IVdsAsync Methods 110
3.3.5.2.2.1 IVdsAsync::Cancel (Opnum 3) 110
3.3.5.2.2.2 IVdsAsync::Wait (Opnum 4) 111
3.3.5.2.2.3 IVdsAsync::QueryStatus (Opnum 5) 112
3.3.5.2.3 IVdsServiceLoader Methods 112
3.3.5.2.3.1 IVdsServiceLoader::LoadService (Opnum 3) 112
3.3.5.2.4 IVdsService Methods 113
3.3.5.2.4.1 IVdsService::IsServiceReady (Opnum 3) 113
3.3.5.2.4.2 IVdsService::WaitForServiceReady (Opnum 4) 113
3.3.5.2.4.3 IVdsService::GetProperties (Opnum 5) 113
3.3.5.2.4.4 IVdsService::QueryProviders (Opnum 6) 114
3.3.5.2.4.5 IVdsService::QueryUnallocatedDisks (Opnum 8) 114
3.3.5.2.4.6 IVdsService::GetObject (Opnum 9) 115
3.3.5.2.4.7 IVdsService::QueryDriveLetters (Opnum 10) 115
3.3.5.2.4.8 IVdsService::QueryFileSystemTypes (Opnum 11) 116
3.3.5.2.4.9 IVdsService::Reenumerate (Opnum 12) 117
3.3.5.2.4.10 IVdsService::Refresh (Opnum 13) 117
3.3.5.2.4.11 IVdsService::CleanupObsoleteMountPoints (Opnum 14) 118
3.3.5.2.4.12 IVdsService::Advise (Opnum 15) 118
3.3.5.2.4.13 IVdsService::Unadvise (Opnum 16) 119
3.3.5.2.4.14 IVdsService::Reboot (Opnum 17) 119
3.3.5.2.4.15 IVdsService::SetFlags (Opnum 18) 119
3.3.5.2.4.16 IVdsService::ClearFlags (Opnum 19) 120
3.3.5.2.5 IVdsServiceInitialization Methods 120
3.3.5.2.5.1 IVdsServiceInitialization::Initialize (Opnum 3) 120
3.3.5.2.6 IVdsServiceUninstallDisk Methods 120
3.3.5.2.6.1 IVdsServiceUninstallDisk::GetDiskIdFromLunInfo (Opnum 3) 120
3.3.5.2.6.2 IVdsServiceUninstallDisk::UninstallDisks (Opnum 4) 121
3.3.5.2.7 IVdsServiceHba Methods 122
3.3.5.2.7.1 IVdsServiceHba::QueryHbaPorts (Opnum 3) 122
3.3.5.2.8 IVdsServiceIscsi Methods 123
3.3.5.2.8.1 IVdsServiceIscsi::GetInitiatorName (Opnum 3) 123
3.3.5.2.8.2 IVdsServiceIscsi::QueryInitiatorAdapters (Opnum 4) 123
3.3.5.2.8.3 IVdsServiceIscsi::SetInitiatorSharedSecret (Opnum 8) 124
3.3.5.2.9 IVdsHbaPort Methods 124
3.3.5.2.9.1 IVdsHbaPort::GetProperties (Opnum 3) 124
3.3.5.2.9.2 IVdsHbaPort::SetAllPathStatuses (Opnum 4) 125
3.3.5.2.10 IVdsIscsiInitiatorAdapter Methods 125
3.3.5.2.10.1 IVdsIscsiInitiatorAdapter::GetProperties (Opnum 3) 125

3.3.5.2.10.2 IVdsIscsiInitiatorAdapter::QueryInitiatorPortals (Opnum 4) 125
3.3.5.2.11 IVdsIscsiInitiatorPortal Methods 126
3.3.5.2.11.1 IVdsIscsiInitiatorPortal::GetProperties (Opnum 3) 126
3.3.5.2.11.2 IVdsIscsiInitiatorPortal::GetInitiatorAdapter (Opnum 4) 126
3.3.5.2.12 IVdsProvider Methods 127
3.3.5.2.12.1 IVdsProvider::GetProperties (Opnum 3) 127
3.3.5.2.13 IVdsSwProvider Methods 127
3.3.5.2.13.1 IVdsSwProvider::QueryPacks (Opnum 3) 127
3.3.5.2.13.2 IVdsSwProvider::CreatePack (Opnum 4) 128
3.3.5.2.14 IVdsHwProvider Methods 128
3.3.5.2.14.1 IVdsHwProvider::QuerySubSystems (Opnum 3) 128
3.3.5.2.15 IVdsSubSystemImportTarget Methods 129
3.3.5.2.15.1 IVdsSubSystemImportTarget::GetImportTarget (Opnum 3) 129
3.3.5.2.15.2 IVdsSubSystemImportTarget::SetImportTarget (Opnum 4) 129
3.3.5.2.16 IVdsPack Methods 130
3.3.5.2.16.1 IVdsPack::GetProperties (Opnum 3) 130
3.3.5.2.16.2 IVdsPack::GetProvider (Opnum 4) 130
3.3.5.2.16.3 IVdsPack::QueryVolumes (Opnum 5) 131
3.3.5.2.16.4 IVdsPack::QueryDisks (Opnum 6) 131
3.3.5.2.16.5 IVdsPack::CreateVolume (Opnum 7) 132
3.3.5.2.16.6 IVdsPack::AddDisk (Opnum 8) 133
3.3.5.2.16.7 IVdsPack::MigrateDisks (Opnum 9) 134
3.3.5.2.16.8 IVdsPack::RemoveMissingDisk (Opnum 11) 135
3.3.5.2.16.9 IVdsPack::Recover (Opnum 12) 135
3.3.5.2.17 IVdsPack2 Methods 136
3.3.5.2.17.1 IVdsPack2::CreateVolume2 (Opnum 3) 136
3.3.5.2.18 IVdsDisk Methods 137
3.3.5.2.18.1 IVdsDisk::GetProperties (Opnum 3) 137
3.3.5.2.18.2 IVdsDisk::GetPack (Opnum 4) 137
3.3.5.2.18.3 IVdsDisk::GetIdentificationData (Opnum 5) 138
3.3.5.2.18.4 IVdsDisk::QueryExtents (Opnum 6) 138
3.3.5.2.18.5 IVdsDisk::ConvertStyle (Opnum 7) 139
3.3.5.2.18.6 IVdsDisk::SetFlags (Opnum 8) 139
3.3.5.2.18.7 IVdsDisk::ClearFlags (Opnum 9) 140
3.3.5.2.19 IVdsDisk2 Methods 140
3.3.5.2.19.1 IVdsDisk2::SetSANMode (Opnum 3) 140
3.3.5.2.20 IVdsAdvancedDisk Methods 141
3.3.5.2.20.1 IVdsAdvancedDisk::GetPartitionProperties (Opnum 3) 141
3.3.5.2.20.2 IVdsAdvancedDisk::QueryPartitions (Opnum 4) 141
3.3.5.2.20.3 IVdsAdvancedDisk::CreatePartition (Opnum 5) 142
3.3.5.2.20.4 IVdsAdvancedDisk::DeletePartition (Opnum 6) 144
3.3.5.2.20.5 IVdsAdvancedDisk::ChangeAttributes (Opnum 7) 145
3.3.5.2.20.6 IVdsAdvancedDisk::AssignDriveLetter (Opnum 8) 146
3.3.5.2.20.7 IVdsAdvancedDisk::DeleteDriveLetter (Opnum 9) 147
3.3.5.2.20.8 IVdsAdvancedDisk::GetDriveLetter (Opnum 10) 147
3.3.5.2.20.9 IVdsAdvancedDisk::FormatPartition (Opnum 11) 148
3.3.5.2.20.10 IVdsAdvancedDisk::Clean (Opnum 12) 149
3.3.5.2.21 IVdsAdvancedDisk2 Methods 151
3.3.5.2.21.1 IVdsAdvancedDisk2::ChangePartitionType (Opnum 3) 151
3.3.5.2.22 IVdsCreatePartitionEx Methods 152
3.3.5.2.22.1 IVdsCreatePartitionEx::CreatePartitionEx (Opnum 3) 152
3.3.5.2.23 IVdsDiskPartitionMF Methods 154
3.3.5.2.23.1 IVdsDiskPartitionMF::GetPartitionFileSystemProperties (Opnum 3) 154
3.3.5.2.23.2 IVdsDiskPartitionMF::GetPartitionFileSystemTypeName (Opnum 4) 154

3.3.5.2.23.3 IVdsDiskPartitionMF::QueryPartitionFileSystemFormatSupport (Opnum 5) 155
3.3.5.2.23.4 IVdsDiskPartitionMF::FormatPartitionEx (Opnum 6) 155
3.3.5.2.24 IVdsRemovable Methods 157
3.3.5.2.24.1 IVdsRemovable::QueryMedia (Opnum 3) 157
3.3.5.2.24.2 IVdsRemovable::Eject (Opnum 4) 158
3.3.5.2.25 IVdsVolume Methods 158
3.3.5.2.25.1 IVdsVolume::GetProperties (Opnum 3) 158
3.3.5.2.25.2 IVdsVolume::GetPack (Opnum 4) 159
3.3.5.2.25.3 IVdsVolume::QueryPlexes (Opnum 5) 159
3.3.5.2.25.4 IVdsVolume::Extend (Opnum 6) 160
3.3.5.2.25.5 IVdsVolume::Shrink (Opnum 7) 161
3.3.5.2.25.6 IVdsVolume::AddPlex (Opnum 8) 162
3.3.5.2.25.7 IVdsVolume::BreakPlex (Opnum 9) 163
3.3.5.2.25.8 IVdsVolume::RemovePlex (Opnum 10) 164
3.3.5.2.25.9 IVdsVolume::Delete (Opnum 11) 165
3.3.5.2.25.10 IVdsVolume::SetFlags (Opnum 12) 166
3.3.5.2.25.11 IVdsVolume::ClearFlags (Opnum 13) 167
3.3.5.2.26 IVdsVolumeMF Methods 168
3.3.5.2.26.1 IVdsVolumeMF::GetFileSystemProperties (Opnum 3) 168
3.3.5.2.26.2 IVdsVolumeMF::Format (Opnum 4) 169
3.3.5.2.26.3 IVdsVolumeMF::AddAccessPath (Opnum 5) 170
3.3.5.2.26.4 IVdsVolumeMF::QueryAccessPaths (Opnum 6) 171
3.3.5.2.26.5 IVdsVolumeMF::QueryReparsePoints (Opnum 7) 172
3.3.5.2.26.6 IVdsVolumeMF::DeleteAccessPath (Opnum 8) 172
3.3.5.2.26.7 IVdsVolumeMF::Mount (Opnum 9) 173
3.3.5.2.26.8 IVdsVolumeMF::Dismount (Opnum 10) 174
3.3.5.2.26.9 IVdsVolumeMF::SetFileSystemFlags (Opnum 11) 174
3.3.5.2.26.10 IVdsVolumeMF::ClearFileSystemFlags (Opnum 12) 175
3.3.5.2.27 IVdsVolumeMF2 Methods 175
3.3.5.2.27.1 IVdsVolumeMF2::GetFileSystemTypeName (Opnum 3) 175
3.3.5.2.27.2 IVdsVolumeMF2::QueryFileSystemFormatSupport (Opnum 4) 175
3.3.5.2.27.3 IVdsVolumeMF2::FormatEx (Opnum 5) 176
3.3.5.2.28 IVdsVolumeShrink Methods 178
3.3.5.2.28.1 IVdsVolumeShrink::QueryMaxReclaimableBytes (Opnum 3) 178
3.3.5.2.28.2 IVdsVolumeShrink::Shrink (Opnum 4) 178
3.3.5.2.29 IVdsVolumeOnline Methods 180
3.3.5.2.29.1 IVdsVolumeOnline::Online (Opnum 3) 180
3.3.5.2.30 IVdsVolumePlex Methods 180
3.3.5.2.30.1 IVdsVolumePlex::GetProperties (Opnum 3) 180
3.3.5.2.30.2 IVdsVolumePlex::GetVolume (Opnum 4) 180
3.3.5.2.30.3 IVdsVolumePlex::QueryExtents (Opnum 5) 181
3.3.5.2.30.4 IVdsVolumePlex::Repair (Opnum 6) 181
3.3.6 Timer Events 183
3.3.7 Other Local Events 183
3.3.7.1 Disk Pack Arrival (Dynamic Disks) 183
3.3.7.2 Disk Pack Removal (Dynamic Disks) 183
3.3.7.3 Disk Arrival 183
3.3.7.4 Disk Removal 183
3.3.7.5 Volume Arrival 183
3.3.7.6 Volume Removal 183
3.3.7.7 File System Modification 183
3.3.7.8 Mount Point Change 184
3.3.7.9 Drive Letter Assignment 184
3.3.7.10 Drive Letter Removal 184

3.3.7.11 Media Arrival 185
3.3.7.12 Media Removal 185

4 Protocol Examples 186
4.1 VDS Sessions 186
4.1.1 Starting Sessions 186
4.1.2 Ending Sessions 189
4.2 VDS Client Notifications 189
4.2.1 Registering for Notifications 189
4.2.2 Receiving Notifications 190
4.2.3 Unregistering for Notifications 190
4.3 Querying Enumerations of VDS Objects 191
4.4 Retrieving the Properties and IDs of VDS Objects 192
4.5 Performing Asynchronous Tasks 193

5 Security 196
5.1 Security Considerations for Implementers 196
5.2 Index of Security Parameters 196

6 Appendix A: Full IDL 197

7 Appendix B: Windows Behavior 230

8 Index 240

1 Introduction

The Virtual Disk Service (VDS) Protocol is a set of **distributed component object model (DCOM) interfaces** for managing the configuration of **disk** storage on a computer. The VDS Protocol deals with detailed low-level operating system and storage concepts.

Although this specification outlines the basic concepts that you need to know, this specification assumes that you are familiar with these technologies. For information about storage, disk, and **volume** concepts, see [MSDN-STC] and [MSDN-PARTITIONINFO]; for information on disk management, see [MSDN-DISKMAN]. For more information about programming VDS, see [MSDN-VDSPG].

The VDS Protocol is used to programmatically enumerate and configure disks, volumes, **host bus adapter (HBA)** ports, and **iSCSI initiators** on local and remote computers that run the Windows Server 2003 operating system and later. This protocol supersedes the Disk Management Remote Protocol, as specified in [MS-DMRP], that is used in the Windows 2000 and Windows XP operating systems.

1.1 Glossary

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

Allocation Unit Size
Basic Disk
Basic Provider
Basic Volume
BitLocker
Boot Configuration File
Boot Loader
Boot Partition
Boot Volume
Bus
Challenge Handshake Authentication Protocol (CHAP)
Cluster
Cluster Size
Compact Disc File System (CDFS)
Component Object Model (COM)
Crash Dump File
Cylinder
Disk
Disk Controller
Disk Extent
Disk Group
Disk Group Import
Disk Management Remote Protocol
Disk Pack
Disk Regions
Disk Signature
Disk Type
Distributed Component Object Model (DCOM)
Drive Letter
Dynamic Disk
Dynamic Provider
Dynamic Volume
Extended Partition

Extensible Firmware Interface (EFI)
FAT File System
FAT32 File System
Fault-Tolerant
Fiber Channel Bus
File Allocation Units
File System Extension
File System Label
Foreign
Format
Free Space
Full Format
Globally Unique Identifier (GUID)
GUID Partitioning Table (GPT)
Hard Disk
Hibernation Image
Host Bus Adapter (HBA)
HRESULT
Interface
Interface Definition Language (IDL)
Internet SCSI (iSCSI)
Logical Disk Manager (LDM)
Logical Drive
Logical Partition
Logical Unit Number (LUN)
Masked Disk
Master Boot Record (MBR)
Merge Disks Or Disk Groups
Microsoft Interface Definition Language (MIDL)
Mirrored Volume
Multipartition Volume
NTFS
Offline
Online
Opnum
Optical Media Drive
Pack
Page File
Partition
Partition Table
Path
Plex
Primary Disk Group
Primary Partition
Quick Format
RAID
RAID-0
RAID-1
RAID-5
RAID Column
Read-Only
Region
Remote Procedure Call (RPC)
Removable Media
Reparse Point

RPC Protocol Sequence
SCSI Logical Unit Number (LUN)
SCSI Port Number
Sector
Shadow Copy
Signature
Simple Volume
Small Computer System Interface (SCSI) Bus
Snapshot
System Partition
Track
Unallocated Disk
Universal Disk Format (UDF)
Universal Serial Bus (USB)
Universally Unique Identifier (UUID)
Unmasked Disk
VDS Provider
VDS Session
Volume
Volume Data
Volume Label
Volume Manager
Volume Plex

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**.

Active Volume: For **volumes** that consist of single **partitions**, **active volume** is synonymous with **active partition**. For **volumes** that consist of multiple **partitions**, **active volume** refers to a **volume** in which one of the **partitions** is an **active partition** (generally mirrored **volumes** because **partitions** on striped or **RAID-5 volumes** do not have complete copies of **volume** data). See also **active partition**.

Disk Quorum: The minimum number of **disks** in a **disk group** that is required to enable the online status of a **disk group**. A **disk quorum** is defined as n/2 + 1, where n is the total number of **disks** in the group. A **disk quorum** prevents **disk groups** from gaining **online** status on more than one computer.

EUI-64: The IEEE-defined 64-bit extended unique identifier (EUI-64). EUI-64 is a concatenation of the 24-bit company_id value by the IEEE Registration Authority and a 40-bit extension identifier that is assigned by the organization with that company_id assignment. For more information, see [EUI64].

File System: A set of data structures for naming, organizing, and storing files in a **volume**. **NTFS**, **FAT**, and **FAT32** are examples of **file system** types.

Import target: An iSCSI target with which the **LUNs** being imported to the subsystem are associated.

iSCSI Initiator: A client of a SCSI interface. An **iSCSI initiator** issues SCSI commands to request services from components, which are logical units of a server known as a "target." For more information, see [RFC3720] section 1.

iSCSI Initiator Adapter: The hardware that allows an **iSCSI initiator** to communicate with other computers on the network. For more information, see [RFC3720] section 9.1.

iSCSI Initiator Portal: The component of an **iSCSI initiator** that has a TCP/IP network address and that may be used by an iSCSI node in that network entity for the connections in one of its iSCSI sessions. For more information, see [RFC3720] section 3.4.

iSCSI Session: A group of TCP connections that link an **iSCSI initiator** with a target. For more information, see [RFC3720] section 3.4.

iSCSI Target: A server of a SCSI interface and also a logical unit of a server that responds to SCSI command requests from an iSCSI initiator. A target device contains one or more SCSI target ports and one or more device servers and associated logical units. For more information, see [RFC3720] section 1.

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

Partition Type: A value that indicates the intended use of the **partition** or the type of **file system** on the **partition**. For an **MBR**-formatted **disk**, the type is a single byte value. For a **GPT**-formatted **disk**, the type is a **GUID**. For example, on **MBR disks**, **partition** type 0x07 indicates that the **partition** is formatted with the **NTFS** file system. Original equipment manufacturers (OEMs) typically designate a **partition** type of 0x12 to indicate that manufacturer-specific data is stored on the **partition**.

SCSI Name String Identifier: An identifier string that is used to identify a **SCSI bus** device. For more information, see [SPC-3].

Shared Secret: A symmetric encryption key that is shared by two entities, such as a user and a domain controller, and that has a long lifetime. A password is a common example of a **shared secret**. Also called a "secret key".

Striped Volume: See **RAID-0**.

Subsystem: A storage device that coordinates and controls the operation of one or more disk drives.

System Volume: For **volumes** that consist of single **partitions**, **system volume** is synonymous with **system partition**. For **volumes** that consist of multiple **partitions**, **system volume** refers to a **volume** in which one of the **partitions** is a system **partition** (generally **mirrored volumes**, because **partitions** on striped or **RAID-5 volumes** do not have complete copies of **volume** data). See also **system partition**.

Virtual Disk Service (VDS): If the term is used as a noun, **VDS** refers to the service component that runs on the server. If **VDS** is used as an adjective, it refers to the protocol that is specified in this document (which the service uses to communicate with clients).

VDS Object: An instance of a class that exposes one or more **DCOM interfaces** to query or configure the **VDS** service, the operating system device (such as a **disk** or **volume**), or the concept (such as a software provider) that the object represents. Each object has an

associated type that indicates the type of device or concept that it represents. Unless otherwise indicated, the term "object" refers to a **VDS** object.

Volume Plex Member: A **RAID** construct for organizing **disks** and **volumes**. Also called a **RAID column**.

Windows Preinstallation Environment (Windows PE): A minimal Windows system environment that provides limited services based on the Windows XP, Windows Server 2003, or Windows Vista kernels. It provides the minimum set of features that are required to run the operating system setup, perform system recovery, access and install operating systems from the network, script basic repetitive tasks, and validate hardware.

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

1.2 References

1.2.1 Normative References

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

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

[EUI64] IEEE Standards Association, "Guidelines for 64-bit Global Identifier (EUI-64) Registration Authority", http://standards.ieee.org/regauth/oui/tutorials/EUI64.html

[HBAAPI] Storage Networking Industry Association, "Common HBA API", T11 Document 02-149v0, March 2002, ftp://ftp.t11.org/t11/docs/02-149v0.pdf

[IEC60908] International Electrotechnical Commission, "Audio Recording - Compact Disc Digital Audio System", IEC 60908 Ed. 2.0, 1999.

If you have any trouble finding [IEC60908], please check here.

[MS-CHAP] Microsoft Corporation, "Extensible Authentication Protocol Method for Microsoft Challenge Handshake Authentication Protocol (CHAP) Specification", January 2007.

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

[MS-DMRP] Microsoft Corporation, "Disk Management Remote Protocol Specification", August 2007.

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

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

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

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

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

[RFC3720] Satran, J., et al., "Internet Small Computer Systems Interface (iSCSI)", RFC 3720, April 2004, http://www.ietf.org/rfc/rfc3720.txt

[SPC-3] International Committee on Information Technology Standards, "SCSI Primary Commands - 3 (SPC-3)", Project T10/1416-D, May 2005, http://www.t10.org/ftp/t10/drafts/spc3/spc3r23.pdf

1.2.2 Informative References

[MSDN-DATA] Microsoft Corporation, "Windows Data Types", http://msdn2.microsoft.com/en-us/library/aa383751.aspx

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

[MSDN-PARTITIONINFO] Microsoft Corporation, "PARTITION_INFORMATION_EX", http://msdn2.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-VDSPG] Microsoft Corporation, "Virtual Disk Service Programming Guide", http://msdn2.microsoft.com/en-us/library/aa383063.aspx

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

[MSFT-DISKMANRESKIT] Microsoft Corporation, "Managing Disks and Volumes", January 2005, http://technet2.microsoft.com/WindowsServer/f/?en/Library/f2428fbd-d474-4c8f-904 9-6634b37d805d1033.mspx

[MSFT-XPSP2SEC] Microsoft Corporation, "Changes to Functionality in Microsoft Windows XP Service Pack 2", November 2004, http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/sp2netwk.mspx#EJAA

1.3 Protocol Overview (Synopsis)

The VDS Protocol provides a mechanism for remote configuration of disks, **partitions**, volumes, and iSCSI initiators on a server. Through the VDS Protocol, a client can change the configuration of disks into partitions, partitions into volumes, and volumes into **file systems**. The protocol also enables clients to obtain notifications of changes to these storage objects.

In the VDS Protocol, two entities are involved: the server, whose storage is configured, and the client, which accesses and requests changes to the server storage configuration.

The VDS Protocol is expressed as a set of DCOM interfaces. For a server, this protocol implements support for the DCOM interface in order to manage storage. For a client, this protocol invokes method calls on the interface in order to perform disk and volume configuration tasks on the server.<1>

1.3.1 Method Invocation

1.3.1.1 Hierarchy Overview

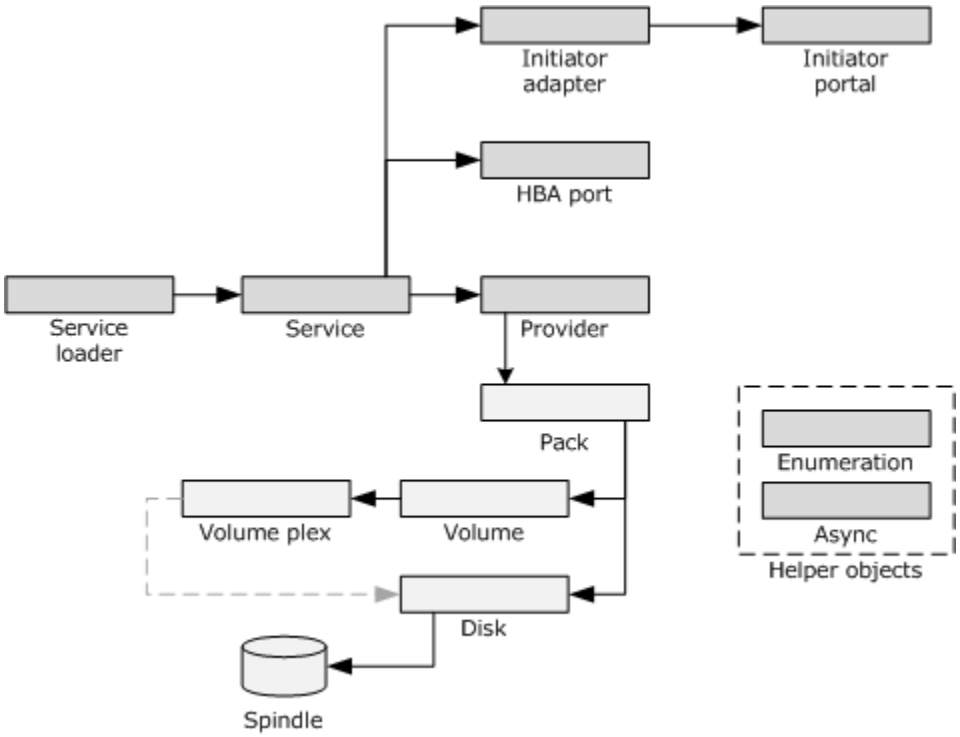

Figure 1: Hierarchy overview

1.3.1.2 Method Sequencing Requirements

Some method calls require no prerequisite calls against the server; they simply query for information or pass in parameters that are constructed by the client. The following listed calls, however, are made in sequence.

In general, the prerequisite call is to an object enumeration method, which retrieves information about a specific set of storage objects, such as volumes or disks. Information that the object enumeration method returns is then used to supply input parameters for subsequent calls. Calls with such prerequisites are grouped in the next topic by storage object type.

1.3.1.3 Storage Object Relationships

This section describes the hierarchy of interfaces and objects that the VDS Protocol uses and the relationships between those objects.

Service Loader and Service: The first interface the client obtains is the **IVdsServiceLoader** interface. The client invokes **IVdsServiceLoader::LoadService** to load the **VDS** service on the specified machine. The server MUST respond with an **IVdsService** interface for the VDS service that is loaded.

Service and Providers: The client invokes **IVdsService::QueryProviders** to obtain a list of providers. The server MUST respond with an **IEnumVdsObject** interface, which enumerates a list of

IUnknown interfaces, one for each provider that is available on the server. The client invokes **QueryInterface** on the **IUnknown** interface to retrieve an **IVdsSwProvider** or **IVdsProvider** interface on the provider object.

Service and Subsystems: The client invokes **IVdsService::QueryProviders** with the flag VDS_QUERY_HARDWARE_PROVIDERS to obtain a list of VDS hardware providers. The server responds with an **IEnumVdsObject** interface, which enumerates a list of **IUnknown** interfaces, one for each hardware provider that is available on the server. The client invokes **QueryInterface** on the **IUnknown** interface to retrieve an **IVdsHwProvider** interface on the provider object. The client invokes **IVdsHwProvider::QuerySubSystems** to obtain a list of subsystems. The server responds with an **IEnumVdsObject** interface, which enumerates a list of **IUnknown** interfaces, one for each subsystem that is available on the server. The client invokes **QueryInterface** on the **IUnknown** interface to retrieve an **IVdsSubSystemImportTarget** interface.

Service and **Unallocated Disks**: The client invokes **IVdsService::QueryUnallocatedDisks** to obtain a list of disks that do not have a recognized disk partitioning **format**. The server MUST respond with an **IEnumVdsObject** interface, which enumerates a list of **IUnknown** interfaces, one for each unallocated disk that is available on the server. The client invokes **IUnknown::QueryInterface** to retrieve an **IVdsDisk**, **IVdsDisk2**, **IVdsAdvancedDisk**, **IVdsAdvancedDisk2**, **IVdsDiskPartitionMF**, **IVdsCreatePartitionEx**, or **IVdsRemovable** interface on the disk object.

Service and File System Types: The client invokes **IVdsService::QueryFileSystemTypes** to obtain a list of the file system types that are available for use in formatting volumes. The server MUST return a list of **VDS_FILE_SYSTEM_TYPE_PROP** structures.

Service and **Drive Letters**: The client invokes **IVdsService::QueryDriveLetters** to obtain a list of drive letters on the system. The server MUST return a list of **VDS_DRIVE_LETTER_PROP** structures; the **bUsed** member indicates whether the drive letter is in use.

Service and HBA Ports: The client invokes **IUnknown::QueryInterface** on the **IVdsService** interface to retrieve the **IVdsServiceHba** interface. The client invokes **IVdsServiceHba::QueryHbaPorts** to obtain a list of the HBA ports that are connected to the server. The server MUST respond with an **IEnumVdsObject** interface, which enumerates a list of **IUnknown** interfaces, one for each HBA port that is connected to the machine. The client invokes **IUnknown::QueryInterface** to retrieve an **IVdsServiceHba** interface on the HBA port object.

Service and Initiator Adapters: The client invokes **IUnknown::QueryInterface** on the **IVdsService** interface to retrieve the **IVdsServiceIscsi** interface. The client invokes **IVdsServiceIscsi::QueryInitiatorAdapters** to obtain a list of the **iSCSI initiator adapters** that are connected to the server. The server MUST respond with an **IEnumVdsObject** interface, which enumerates a list of **IUnknown** interfaces, one for each initiator adapter that is connected to the machine. The client invokes **IUnknown::QueryInterface** to retrieve an **IVdsIscsiInitiatorAdapter** interface on the initiator adapter object.

Service and Initiator Portals: The client invokes **IVdsIscsiInitiatorAdapter::QueryInitiatorPortals** to obtain a list of the **iSCSI initiator portals** that the initiator adapter maintains. The server MUST respond with an **IEnumVdsObject** interface, which enumerates a list of **IUnknown** interfaces, one for each initiator portal. The client invokes **IUnknown::QueryInterface** to retrieve an **IVdsIscsiInitiatorPortal** interface on the initiator portal object.

Providers and Packs: The client invokes **IVdsSwProvider::QueryPacks** to obtain a list of the **packs** that the provider maintains. The server MUST respond with an **IEnumVdsObject** interface, which enumerates a list of **IUnknown** interfaces, one for each pack. The client invokes **IUnknown::QueryInterface** to retrieve an **IVdsPack** or **IVdsPack2** interface on the pack object.

Packs and disks: The client invokes **IVdsPack::QueryDisks** to obtain a list of the disks in the pack. The server MUST respond with an **IEnumVdsObject** interface, which enumerates a list of **IUnknown** interfaces, one for each disk in the pack. The client invokes **IUnknown::QueryInterface** to retrieve an **IVdsDisk**, **IVdsDisk2**, **IVdsAdvancedDisk**, **IVdsAdvancedDisk2**, **IVdsDiskPartitionMF**, **IVdsCreatePartitionEx**, or **IVdsRemovable** interface on the disk object.

Packs and Volumes: The client invokes **IVdsPack::QueryVolumes** to obtain a list of the volumes in the pack. The server MUST respond with an **IEnumVdsObject** interface, which enumerates a list of **IUnknown** interfaces, one for each volume in the pack. The client invokes **IUnknown::QueryInterface** to retrieve an **IVdsVolume**, **IVdsVolume2**, or **IVdsVolumeOnline** interface on the volume object.

Volumes and **Plexes**: The client invokes **IVdsVolume::QueryPlexes** to obtain a list of the plexes for a volume. The server MUST respond with an **IEnumVdsObject** interface, which enumerates a list of **IUnknown** interfaces, one for each plex that is associated with the volume. The client invokes **IUnknown::QueryInterface** to retrieve an **IVdsVolumePlex** interface on the plex object.

Plexes and Extents: The client invokes **IVdsVolumePlex::QueryExtents** to obtain a list of the extents for a specified plex. The server MUST return a list of **VDS_DISK_EXTENT** structures, one for each extent in use by the plex.

Volumes and Drive Letters: The client invokes **IVdsService::QueryDriveLetters** to obtain a list of drive letters on the system. The server MUST return a list of **VDS_DRIVE_LETTER_PROP** structures; the **volumeId** member indicates the volume that is associated with the drive letter.

Volumes and **Reparse Points**: For the **IVdsVolume** interface, the client invokes **IUnknown::QueryInterface** to retrieve an **IVdsVolumeMF** interface. The client then invokes **IVdsVolumeMF::QueryReparsePoints** to obtain a list of **mount points** on the volume. The server MUST return a list of **VDS_REPARSE_POINT_PROP** structures; the **SourceVolumeId** member indicates the mounted volume. For example, for drive D mounted to C:\MountD, drive D is the mounted volume.

Volumes and Access Paths: For the **IVdsVolume** interface, the client invokes **IUnknown::QueryInterface** to retrieve an **IVdsVolumeMF** interface. The client then invokes **IVdsVolumeMF::QueryAccessPaths** to obtain a list of user mode **path** names for the volume. The server MUST return a list of drive letters and mount points for the volume. For drive D mounted to C:\MountD, drive D is the mounted volume and C:\MountD is the mount point.

Volumes and Supported File System Formats: For the **IVdsVolume** interface, the client invokes **IUnknown::QueryInterface** to retrieve an **IVdsVolumeMF2** interface. The client invokes **IVdsVolumeMF2::QueryFileSystemFormatSupport** to obtain a list of file systems that are supported for the volume. The server MUST return a list of **VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP** structures, one for each file system that is supported on the volume.

Disks and Extents: The client invokes **IVdsDisk::QueryExtents** to obtain a list of the extents for a specified disk. The server MUST return a list of **VDS_DISK_EXTENT** structures, one for each extent on the disk.

For a **VDS_DISK_EXTENT** that describes a **disk extent**, a client maps the extent to its disk by obtaining a list of **VDS_DISK_PROP** structures. The client obtains this list by invoking **IVdsPack::QueryDisks** followed by **IVdsDisk::GetProperties** for each disk. The server MUST return a **VDS_DISK_PROP** structure from **IVdsDisk::GetProperties**. The client matches the **VDS_DISK_EXTENT::diskId** member to the **VDS_DISK_PROP::id** member.

Disks and Partitions: For the **IVdsDisk** interface, the client invokes **IUnknown::QueryInterface** to obtain the **IVdsAdvancedDisk** interface. The client invokes **IVdsAdvancedDisk::QueryPartitions** to obtain a list of the partitions for a specified disk. The server MUST return a list of **VDS_PARTITION_PROP** structures, one for each partition on the disk.

Extents and Volumes: For a **VDS_DISK_EXTENT** that describes a disk extent, a client maps the extent to its volume by obtaining a list of **VDS_VOLUME_PROP** structures. The client obtains this list by invoking **IVdsPack::QueryVolumes**, followed by **IVdsVolume::GetProperties** for each volume. The server MUST return a **VDS_VOLUME_PROP** structure from **IVdsVolume::GetProperties**. The client matches the **VDS_DISK_EXTENT::volumeId** member to the **VDS_VOLUME_PROP::id** member.

Extents and Plexes: For a **VDS_DISK_EXTENT** that describes a disk extent, a client maps the extent to its **volume plex** by obtaining a list of **VDS_VOLUME_PLEX_PROP** structures. The client obtains this list by invoking **IVdsVolume::QueryPlexes**, followed by **IVdsVolumePlex::GetProperties** for each plex. The server MUST return a **VDS_VOLUME_PLEX_PROP** structure from **IVdsVolumePlex::GetProperties**. The client matches the **VDS_DISK_EXTENT::plexId** member to the **VDS_VOLUME_PLEX_PROP::id** member.

Volumes and File Systems: For a **VDS_FILE_SYSTEM_PROP** structure that describes a file system, a client maps the file system to a volume by obtaining a list of **VDS_VOLUME_PROP** structures. The client invokes **IVdsPack::QueryVolumes**, followed by **IVdsVolume::GetProperties** for each volume. The server MUST return a **VDS_VOLUME_PROP** structure from **IVdsVolume::GetProperties**. The client matches the **VDS_FILE_SYSTEM_PROP::volumeId** member to the **VDS_VOLUME_PROP::id** member.

Volumes and Drive Letters: For a **VDS_DRIVE_LETTER_PROP** structure that describes a drive letter, a client maps the drive letter to a volume by obtaining a list of **VDS_VOLUME_PROP** structures. The client invokes **IVdsPack::QueryVolumes**, followed by **IVdsVolume::GetProperties** for each volume. The server MUST return a **VDS_VOLUME_PROP** structure from **IVdsVolume::GetProperties**. The client matches the **VDS_DRIVE_LETTER_PROP::volumeId** member to the **VDS_VOLUME_PROP::id** member.

1.3.2 Service and Providers

IVdsService::GetObject—"XXX" is a placeholder for Provider, Pack, Disk, Volume, Volume Plex, or HbaPort. Prior to invoking **GetObject**, the client MUST invoke **QueryXXXs** on interfaces that have a **QueryXXXs** method. The server MUST respond with an **IEnumVdsObject** interface, which enumerates a list of IUnknown interfaces, one for each object that is associated with the enumeration. The client invokes **IUnknown::QueryInterface** to retrieve an **IVdsXXX** interface on the object. The client invokes **IVdsXXX::GetProperties** to retrieve the object ID. The client MUST pass this returned value as the *id* input parameter to the **GetObject** method. **IVdsXXX::GetProperties** returns this value as the **VDS_PROVIDER_PROP**::**id**, **VDS_PACK_PROP**::**id**, **VDS_DISK_PROP**::**id**, **VDS_VOLUME_PROP**::**id**, **VDS_VOLUME_PLEX_PROP**::**id**, or **VDS_HBAPORT_PROP**::**id** structure member. The client can cache the object IDs for the lifetime of the object; it can later be used to retrieve an interface to the object without having to cache the interface to the object itself, or having to enumerate and find the object every time it needs the object.

IVdsService::Advise—Prior to invoking **Advise**, the client MUST call **IVdsService::WaitForServiceReady** or poll by using **IVdsService::IsServiceReady** successfully. The client MUST invoke **IVdsService::Advise** to retrieve the client identification value parameter. The client MUST pass this returned value as the *pdwCookie* input parameter to the **Unadvise** method. **IVdsService::Advise** returns this value as the *pdwCookie* output parameter.

IVdsService::Unadvise—Prior to invoking **Unadvise**, the client MUST invoke **IVdsService::Advise** to retrieve the client identification value parameter. The client MUST pass this returned value as the *pdwCookie* input parameter to the **Unadvise** method. **IVdsService::Advise** returns this value as the *pdwCookie* output parameter.

1.3.3 Packs

IVdsPack::CreateVolume—Prior to invoking **CreateVolume**, the client MUST invoke **IVdsDisk::GetProperties** to retrieve the disk ID parameter. The client MUST pass this returned value as the **VDS_INPUT_DISK**::**diskId** input parameter to the **CreateVolume** method. **CreateVolume** takes an array of one or more **VDS_INPUT_DISK** structures, and **IVdsDisk::GetProperties** MUST be called once for each disk in this array. **IVdsDisk::GetProperties** returns this value as the **VDS_DISK_PROP**::**id** output parameter.

IVdsPack::AddDisk—Prior to invoking **AddDisk**, the client MUST invoke **IVdsDisk::GetProperties** to retrieve the disk ID parameter. The client MUST pass this returned value as the *DiskId* input parameter to the **AddDisk** method. **IVdsDisk::GetProperties** returns this value as the *id* output parameter.

IVdsPack::MigrateDisks—Prior to invoking **MigrateDisks**, the client MUST invoke **IVdsDisk::GetProperties** for each disk in the input array, to retrieve the list of disk id parameters. The client MUST pass this returned value as the *pDiskArray* input parameter to the **MigrateDisks** method. **IVdsDisk::GetProperties** returns this value as the **VDS_DISK_PROP**::**id** output parameter.

Prior to invoking **MigrateDisks**, the client MUST invoke **IVdsDisk::GetProperties** to retrieve the target pack ID parameter. The client MUST pass this returned value as the *TargetPack* input parameter to the **MigrateDisks** method. **IVdsDisk::GetProperties** returns this value as the **VDS_DISK_PROP**::**id** output parameter.

IVdsPack::ReplaceDisk—Prior to invoking **ReplaceDisk**, the client MUST invoke **IVdsDisk::GetProperties** for both the disk being replaced and the replacement disk, to retrieve the old and new disk ID parameters. The client MUST pass these returned values as the *OldDiskId* and *NewDiskId* input parameters to the **ReplaceDisk** method. **IVdsDisk::GetProperties** returns this value as the **VDS_DISK_PROP**::**id** output parameter.

IVdsPack::RemoveMissingDisk—Prior to invoking **RemoveMissingDisk**, the client MUST invoke **IVdsDisk::GetProperties** to retrieve the disk ID parameter. The client MUST pass this returned value as the *DiskId* input parameter to the **RemoveMissingDisk** method. **IVdsDisk::GetProperties** returns this value as the **VDS_DISK_PROP**::**id** output parameter.

IVdsPack2::CreateVolume2—**CreateVolume2** has the same call sequence description as **IVdsPack::CreateVolume**.

1.3.4 Disks

IVdsAdvancedDisk::GetPartitionProperties—Prior to invoking **GetPartitionProperties**, the client MUST invoke **IVdsDisk::QueryExtents** or IVdsVolumePlex::QueryExtents to retrieve the offset parameter. The client MUST pass this returned value as the *ullOffset* input parameter to the **GetPartitionProperties** method. IVdsDisk::QueryExtents or **IVdsVolumePlex::QueryExtents** returns this value as the **VDS_DISK_EXTENT**::**ullOffset** output parameter.

IVdsAdvancedDisk::CreatePartition—Prior to invoking **CreatePartition**, the client MUST invoke **IVdsDisk::QueryExtents** or **IVdsVolumePlex::QueryExtents** to retrieve the free disk extents. Using the list of free disk extents, the client can calculate an offset and size for the new partition. The client MUST pass these calculated values as the *ullOffset* and *ullSize* input parameters to the

CreatePartition method. **IVdsDisk::QueryExtents** and **IVdsVolumePlex::QueryExtents** return a list of **VDS_DISK_EXTENT** structures as an output parameter. These structures contain the offset and size of the free extent as **VDS_DISK_EXTENT**::**ullOffset** and **VDS_DISK_EXTENT**::**ullSize**.

Prior to invoking **CreatePartition**, the client MUST invoke **IVdsAdvancedDisk::GetPartitionProperties** to retrieve the **partition type** for the disk. The client MUST pass this value as the **CREATE_PARTITION_PARAMETERS**::**style** input parameter to the **CreatePartition** method. **IVdsAdvancedDisk**::**GetPartitionProperties** returns this value as the **VDS_PARTITION_PROP**::**PartitionStyle** structure member.

Prior to invoking **CreatePartition**, the client MUST invoke **IVdsDisk**::**AddDisk** to set the partitioning format for the disk. See **IVdsPack**::**IVdsDisk**.

IVdsAdvancedDisk::DeletePartition—Prior to invoking **DeletePartition**, the client MUST invoke **IVdsAdvancedDisk**::**GetPartitionProperties**, **IVdsDisk::QueryExtents**, or **IVdsVolumePlex::QueryExtents** to retrieve the offset parameter. The client MUST pass this returned value as the *ullOffset* input parameter to the **DeletePartition** method. If an invalid offset is passed to this method, it will fail. **IVdsDisk::QueryExtents** or **IVdsVolumePlex::QueryExtents** returns this value as the **VDS_DISK_EXTENT**::**ullOffset** output parameter. **IVdsAdvancedDisk**::**GetPartitionProperties** returns this value as the **VDS_PARTITION_PROP**::**ullOffset** output parameter.

IVdsAdvancedDiskChangeAttributes—**ChangeAttributes** has the same call sequence description as **DeletePartition**.

IVdsAdvancedDiskAssignDriveLetter—**AssignDriveLetter** has the same call sequence description as **DeletePartition**.

IVdsAdvancedDisk::GetDriveLetter—**GetDriveLetter** has the same call sequence description as **DeletePartition**.

IVdsAdvancedDisk::FormatPartition—For call sequencing related to the *ullOffset* input parameter, see the description for IVdsAdvancedDisk::DeletePartition. For call sequencing related to the type input parameter, see sections 1.3.1.3, 3.3.1.4, 3.3.1.5, and 4.5.

IVdsAdvancedDisk2::ChangePartitionType—For call sequencing related to the *ullOffset* input parameter, see the description for IVdsAdvancedDisk::DeletePartition.

Prior to invoking **ChangePartitionType**, the client MUST invoke IVdsAdvancedDisk::GetPartitionProperties to retrieve the partition type for the disk. The client MUST pass this value as the **CHANGE_PARTITION_TYPE_PARAMETERS**::**style** input parameter to the **ChangePartitionType** method. IVdsAdvancedDisk::GetPartitionProperties returns this value as the **VDS_PARTITION_PROP**::**PartitionStyle** structure member.

IVdsCreatePartitionEx::CreatePartitionEx— **IVdsCreatePartitionEx** has the same call sequence description as IVdsAdvancedDisk::CreatePartition.

IVdsServiceUninstallDisk::GetDiskIdFromLunInfo—Prior to invoking **GetDiskIdFromLunInfo**, the client MUST invoke IVdsDisk::GetIdentificationData to retrieve the **logical unit number (LUN)** information for the disk. The client MUST pass this returned value as the *pLunInfo* input parameter to the **GetDiskIdFromLunInfo** method. IVdsDisk::GetIdentificationData returns this value as the *pLunInfo* output parameter.

IVdsServiceUninstallDisk::UninstallDisks—Prior to invoking **UninstallDisks**, the client MUST invoke IVdsDisk::GetProperties for each disk in the input array, to retrieve the list of disk IDs. The client MUST pass this returned value as the *pDiskIdArray* input parameter to the **UninstallDisks** method. IVdsDisk::GetProperties returns this value as the **VDS_DISK_PROP**::**id** output parameter.

1.3.5 Volumes

IVdsVolume::Extend—Prior to invoking **Extend**, the client MUST invoke IVdsDisk::GetProperties for each disk in the input array in order to retrieve the list of disk IDs. The client MUST pass this returned value as the *pInputDiskArray* input parameter to the **Extend** method. IVdsDisk::GetProperties returns this value as the **VDS_DISK_PROP**::**id** output parameter.

IVdsVolume::AddPlex—Prior to invoking **AddPlex**, the client MUST invoke IVdsVolume::GetProperties to retrieve the volume ID. The client MUST pass this returned value as the VolumeId input parameter to the **AddPlex** method. IVdsVolume::GetProperties returns this value as the **VDS_DISK_PROP**::**id** output parameter. For more information, see sections 1.3.1.3, 3.3.1.4, 3.3.1.5, and 4.5.

IVdsVolume::BreakPlex—Prior to invoking **BreakPlex**, the client MUST invoke IVdsVolumePlex::GetProperties to retrieve the plex ID. The client MUST pass this returned value as the *plexId* input parameter to the **BreakPlex** method. IVdsVolumePlex::GetProperties returns this value as the **VDS_DISK_PROP**::**id** output parameter. For more information, see section 1.3.1.3.

IVdsVolume::RemovePlex—**RemovePlex** has the same call sequence description as IVdsVolume::BreakPlex.

IVdsVolumePlex::Repair—**Repair** has the same call sequence description as IVdsVolume::Extend.

1.3.6 File Systems, Drive Letters, and Access Paths

IVdsVolumeMF::Format—For call sequencing related to the *type* input parameter, see "Service and File System Types" in section 1.3.1.3.

IVdsVolumeMF::DeleteAccessPath—For call sequencing related to the *pwszPath* input parameter, see "Volumes and Access Paths," and "Volumes and Drive Letters" in section 1.3.1.3.

IVdsVolumeMF2::FormatEx—Prior to invoking **FormatEx**, the client MUST invoke IVdsDiskPartitionMF::QueryPartitionFileSystemFormatSupport or IVdsVolumeMF2::QueryPartitionFileSystemFormatSupport to retrieve the list of supported file system type names and the associated file system version number. The client MUST pass these returned values as the *pwszFileSystemTypeName* and *usFileSystemRevision* input parameters to the **FormatEx** method. IVdsVolumeMF2::QueryPartitionFileSystemFormatSupport and IVdsVolumeMF2::QueryFileSystemFormatSupport return these values as the **VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP**::**wszName** and **VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP**::**usRevision** output parameters. For more information, see sections 3.3.1.4, 3.3.1.5, and 4.5.

IVdsDiskPartitionMF::GetPartitionFileSystemProperties—**GetPartitionFileSystemProperties** has the same call sequencing description as IVdsAdvancedDisk::DeletePartition.

IVdsDiskPartitionMF::GetPartitionFileSystemTypeName—**GetPartitionFileSystemTypeName** has the same call sequencing description as IVdsAdvancedDisk::DeletePartition.

IVdsDiskPartitionMF::QueryPartitionFileSystemFormatSupport—**QueryPartitionFileSystemFormatSupport** has the same call sequencing description as IVdsAdvancedDisk::DeletePartition.

IVdsDiskPartitionMF::QueryPartitionFileSystemFormatSupport—For call sequencing related to the *ullOffset* input parameter, see the preceding description for IVdsAdvancedDisk::DeletePartition. For call sequencing related to the *pwszFileSystemTypeName* and *usFileSystemRevision* input parameters, see the preceding description for IVdsVolumeMF2::FormatEx.

1.4 Relationship to Other Protocols

The VDS Protocol relies on DCOM, as specified in [MS-DCOM], which uses **remote procedure call (RPC)** as its transport. Beginning with Windows Server 2003, the VDS Protocol replaces the Disk Management Remote Protocol, as specified in [MS-DMRP], for storage management tasks .

1.5 Prerequisites and Preconditions

The VDS Protocol is implemented over DCOM (as specified in DCOM) and RPC; and as a result, has DCOM prerequisites, as specified in [MS-DCOM] and [MS-RPCE], as being common to DCOM and RPC interfaces.

The VDS 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.

An operating system on which an implementation of the VDS Protocol is to run must be able to dynamically requery the list of storage devices and mount points that are available during run time. For more information on these requirements, see sections 3.3.5.2.4.10 and 3.3.5.2.4.11.

1.6 Applicability Statement

The VDS Protocol applies when an application needs to remotely configure disks, volumes, and iSCSI initiators.

You may also use DMRP, as specified in [MS-DMRP], to perform logical functions that are similar to those that the VDS Protocol performs.<2>

1.7 Versioning and Capability Negotiation

Supported Transports: The VDS 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 about transport, see section 2.1.

Protocol Version: The VDS Protocol comprises 29 DCOM interfaces, which are all version 0.0.<3>

Functionality Negotiation: The client negotiates for a specified set of server functionality by specifying the **UUID** that corresponds to the necessary RPC interface by means of **COM** IUnknown::QueryInterface when binding to the server. Certain interfaces are implemented by only particular objects on the server. For more information on storage management objects, see section 3.3.1.2.

Security and Authentication Methods: For more information, see DCOM, as specified in [MS-DCOM], and RPC, as specified in [MS-RPCE].

1.8 Vendor-Extensible Fields

The VDS Protocol does not define any vendor-extensible fields.

1.9 Standards Assignments

The following table shows the Microsoft private assignments for the VDS Protocol.

Parameter	Value	Reference
RPC interface UUID for IEnumVdsObject	118610B7-8D94-4030-B5B8-500889788E4E	None
RPC interface UUID for IVdsAdviseSink	8326CD1D-CF59-4936-B786-5EFC08798E25	None
RPC interface UUID for IVdsAsync	D5D23B6D-5A55-4492-9889-397A3C2D2DBC	None
RPC interface UUID for IVdsServiceLoader	E0393303-90D4-4A97-AB71-E9B671EE2729	None
RPC interface UUID for IVdsService	0818A8EF-9BA9-40D8-A6F9-E22833CC771E	None
RPC interface UUID for IVdsServiceInitialization	4AFC3636-DB01-4052-80C3-03BBCB8D3C69	None
RPC interface UUID for IVdsProvider	10C5E575-7984-4E81-A56B-431F5F92AE42	None
RPC interface UUID for IVdsSwProvider	9AA58360-CE33-4F92-B658-ED24B14425B8	None
RPC interface UUID for IVdsHwProvider	D99BDAAE-B13A-4178-9FDB-E27F16B4603E	None
RPC interface UUID for IVdsSubSystemImportTarget	83BFB87F-43FB-4903-BAA6-127F01029EEC	None
RPC interface UUID for IVdsPack	3B69D7F5-9D94-4648-91CA-79939BA263BF	None
RPC interface UUID for IVdsDisk	07E5C822-F00C-47A1-8FCE-B244DA56FD06	None
RPC interface UUID for IVdsAdvancedDisk	6E6F6B40-977C-4069-BDDD-AC710059F8C0	None
RPC interface UUID for IVdsCreatePartitionEx	9882F547-CFC3-420B-9750-00DFBEC50662	None
RPC interface UUID for IVdsDiskPartitionMF	538684E0-BA3D-4BC0-ACA9-164AFF85C2A9	None
RPC interface UUID for IVdsRemovable	0316560B-5DB4-4ED9-BBB5-213436DDC0D9	None
RPC interface UUID for IVdsVolume	88306BB2-E71F-478C-86A2-79DA200A0F11	None
RPC interface UUID for IVdsVolumeMF	EE2D5DED-6236-4169-931D-B9778CE03DC6	None
RPC interface UUID for IVdsVolumeShrink	D68168C9-82A2-4F85-B6E9-74707C49A58F	None
RPC interface UUID for IVdsVolumeOnline	1BE2275A-B315-4F70-9E44-	None

Parameter	Value	Reference
	879B3A2A53F2	
RPC interface UUID for IVdsVolumePlex	4DAA0135-E1D1-40F1-AAA5-3CC1E53221C3	None
RPC interface UUID for IVdsPack2	13B50BFF-290A-47DD-8558-B7C58DB1A71A	None
RPC interface UUID for IVdsDisk2	40F73C8B-687D-4A13-8D96-3D7F2E683936	None
RPC interface UUID for IVdsAdvancedDisk2	9723F420-9355-42DE-AB66-E31BB15BEEAC	None
RPC interface UUID for IVdsVolumeMF2	4DBCEE9A-6343-4651-B85F-5E75D74D983C	None
RPC interface UUID for IVdsServiceUninstallDisk	B6B22DA8-F903-4BE7-B492-C09D875AC9DA	None
RPC interface UUID for IVdsServiceHba	0AC13689-3134-47C6-A17C-4669216801BE	None
RPC interface UUID for IVdsServiceIscsi	14FBE036-3ED7-4E10-90E9-A5FF991AFF01	None
RPC interface UUID for IVdsHbaPort	2ABD757F-2851-4997-9A13-47D2A885D6CA	None
RPC interface UUID for IVdsIscsiInitiatorAdapter	B07FEDD4-1682-4440-9189-A39B55194DC5	None
RPC interface UUID for IVdsIscsiInitiatorPortal	38A0A9AB-7CC8-4693-AC07-1F28BD03C3DA	None
COM class UUID for service object class	7D1933CB-86F6-4A98-8628-01BE94C9A575	None

2 Messages

The following sections specify how VDS Protocol messages are transported and also specify VDS Protocol message syntax.

2.1 Transport

The VDS Protocol uses 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 defined in [MS-RPCE].

To access an interface, the client requests a DCOM connection to its object UUID endpoint on the server, as specified in section 1.9.

The RPC version number for all interfaces is 0.0.

An implementation of the VDS Protocol MAY configure its DCOM implementation or underlying RPC transport with authentication parameters to restrict client connections. The details of this are implementation-specific.<4>

The VDS Protocol interfaces make use of the underlying DCOM security framework, as specified in [MS-DCOM], and rely on it for access control. DCOM differentiates between launch and access. An implementation of the VDS Protocol MAY differentiate between launch and access permission, and MAY impose different authorization requirements for each interface.<5>

2.2 Message Syntax

The following sections specify VDS Protocol message syntax.<6>

2.2.1 Common Data Types

2.2.1.1 Data Types

2.2.1.1.1 ULONGLONG

The **ULONGLONG** data type is defined in [MS-DTYP].

2.2.1.1.2 DWORD

The **DWORD** data type is defined in [MS-DTYP].

2.2.1.1.3 VDS_OBJECT_ID

This type is declared as follows:

```
typedef GUID VDS_OBJECT_ID;
```

The **VDS_OBJECT_ID** data type defines the **VDS object** identifier as a **GUID** for VDS Protocol storage objects.

2.2.1.1.4 VDS_LUN_INFORMATION

Constant/value	Description
VDS_LUN_INFORMATION 0x00000001	Defines the current version of the VDS_LUN_INFORMATION structure.

2.2.1.2 Enumerations

2.2.1.2.1 VDS_HEALTH

The **VDS_HEALTH** enumeration defines the possible health states of the storage objects in the VDS Protocol. The storage objects are packs, volumes, volume plexes and disks.

```
typedef  enum _VDS_HEALTH
{
  VDS_H_UNKNOWN = 0x00000000,
  VDS_H_HEALTHY = 0x00000001,
  VDS_H_REBUILDING = 0x00000002,
  VDS_H_STALE = 0x00000003,
  VDS_H_FAILING = 0x00000004,
  VDS_H_FAILING_REDUNDANCY = 0x00000005,
  VDS_H_FAILED_REDUNDANCY = 0x00000006,
  VDS_H_FAILED_REDUNDANCY_FAILING = 0x00000007,
  VDS_H_FAILED = 0x00000008
} VDS_HEALTH;
```

VDS_H_UNKNOWN: The health of the object cannot be determined.

VDS_H_HEALTHY: The object indicates **online** status. If the object is a disk, the disk is not missing, log and configuration files are synchronized, and the disk is free of input/output errors. If the object is a LUN or volume, all plexes (mirrored, simple, spanned, and striped) and columns (**RAID-5**) are active.

VDS_H_REBUILDING: The volume is resynchronizing all plexes, or a striped with parity (RAID-5) plex is regenerating the parity.

VDS_H_STALE: The object configuration is stale.

VDS_H_FAILING: The object is failing but still working. For example, a volume with failing health might produce occasional input/output errors from which it can still recover.

VDS_H_FAILING_REDUNDANCY: One or more plexes have errors, but the object is working and all plexes are online.

VDS_H_FAILED_REDUNDANCY: One or more plexes have failed, but at least one plex is working.

VDS_H_FAILED_REDUNDANCY_FAILING: The last working plex is failing.

VDS_H_FAILED: The object has failed. Any object with a failed health status also has a failed object status.

2.2.1.2.2 VDS_NOTIFICATION_TARGET_TYPE

The **VDS_NOTIFICATION_TARGET_TYPE** enumeration defines the set of valid target types (subjects) of a VDS Protocol notification.

```
typedef  enum _VDS_NOTIFICATION_TARGET_TYPE
{
  VDS_NTT_UNKNOWN = 0x00000000,
  VDS_NTT_PACK = 0x0000000A,
  VDS_NTT_VOLUME = 0x0000000B,
  VDS_NTT_DISK = 0x0000000D,
  VDS_NTT_PARTITION = 0x0000003C,
  VDS_NTT_DRIVE_LETTER = 0x0000003D,
  VDS_NTT_FILE_SYSTEM = 0x0000003E,
  VDS_NTT_MOUNT_POINT = 0x0000003F
} VDS_NOTIFICATION_TARGET_TYPE;
```

VDS_NTT_UNKNOWN: Notification is of an unknown type.

VDS_NTT_PACK: Notification refers to a pack.

VDS_NTT_VOLUME: Notification refers to a volume.

VDS_NTT_DISK: Notification refers to a disk.

VDS_NTT_PARTITION: Notification refers to a partition.

VDS_NTT_DRIVE_LETTER: Notification refers to a drive letter.

VDS_NTT_FILE_SYSTEM: Notification refers to a file system.

VDS_NTT_MOUNT_POINT: Notification refers to a mount point.

2.2.1.2.3 VDS_ASYNC_OUTPUT_TYPE

The **VDS_ASYNC_OUTPUT_TYPE** enumeration defines the types of operation information that the **VDS_ASYNC_OUTPUT** structure returns.

```
typedef  enum _VDS_ASYNC_OUTPUT_TYPE
{
  VDS_ASYNCOUT_UNKNOWN = 0x00000000,
  VDS_ASYNCOUT_CREATEVOLUME = 0x00000001,
  VDS_ASYNCOUT_EXTENDVOLUME = 0x00000002,
  VDS_ASYNCOUT_SHRINKVOLUME = 0x00000003,
  VDS_ASYNCOUT_ADDVOLUMEPLEX = 0x00000004,
  VDS_ASYNCOUT_BREAKVOLUMEPLEX = 0x00000005,
  VDS_ASYNCOUT_REMOVEVOLUMEPLEX = 0x00000006,
  VDS_ASYNCOUT_REPAIRVOLUMEPLEX = 0x00000007,
  VDS_ASYNCOUT_RECOVERPACK = 0x00000008,
  VDS_ASYNCOUT_REPLACEDISK = 0x00000009,
  VDS_ASYNCOUT_CREATEPARTITION = 0x0000000A,
  VDS_ASYNCOUT_CLEAN = 0x0000000B,
  VDS_ASYNCOUT_FORMAT = 0x00000065
} VDS_ASYNC_OUTPUT_TYPE;
```

VDS_ASYNCOUT_UNKNOWN: Information is about an unknown type of operation.

VDS_ASYNCOUT_CREATEVOLUME: Information is about creating a volume.

VDS_ASYNCOUT_EXTENDVOLUME: Information is about extending the size of a volume.

VDS_ASYNCOUT_SHRINKVOLUME: Information is about shrinking the size of a volume.

VDS_ASYNCOUT_ADDVOLUMEPLEX: Information is about adding a volume plex.

VDS_ASYNCOUT_BREAKVOLUMEPLEX: Information is about breaking a volume plex.

VDS_ASYNCOUT_REMOVEVOLUMEPLEX: Information is about removing a volume plex.

VDS_ASYNCOUT_REPAIRVOLUMEPLEX: Information is about repairing a volume plex.

VDS_ASYNCOUT_RECOVERPACK: Information is about recovering a pack.

VDS_ASYNCOUT_REPLACEDISK: Information is about replacing a disk.

VDS_ASYNCOUT_CREATEPARTITION: Information is about creating a partition.

VDS_ASYNCOUT_CLEAN: Information is about cleaning a disk.

VDS_ASYNCOUT_FORMAT: Information is about formatting a file system.

2.2.1.2.4 VDS_STORAGE_BUS_TYPE

The **VDS_STORAGE_BUS_TYPE** enumeration defines the type of **bus** on which a disk resides.

```
typedef  enum _VDS_STORAGE_BUS_TYPE
{
  VDSBusTypeUnknown = 0x00000000,
  VDSBusTypeScsi = 0x00000001,
  VDSBusTypeAtapi = 0x00000002,
  VDSBusTypeAta = 0x00000003,
  VDSBusType1394 = 0x00000004,
  VDSBusTypeSSA = 0x00000005,
  VSDBusTypeFibre = 0x00000006,
  VDSBusTypeUsb = 0x00000007,
  VDSBusTypeRAID = 0x00000008,
  VDSBusTypeiScsi = 0x00000009,
  VDSBusTypeSas = 0x0000000A,
  VDSBusTypeSata = 0x0000000B,
  VDSBusTypeSd = 0x0000000C,
  VDSBusTypeMmc = 0x0000000D,
  VDSBusTypeMax = 0x0000000E,
  VDSBusTypeMaxReserved = 0x0000007F
} VDS_STORAGE_BUS_TYPE;
```

VDSBusTypeUnknown: Bus type is unknown.

VDSBusTypeScsi: Disk resides on a **SCSI bus**.

VDSBusTypeAtapi: Disk resides on an AT Attachment Packet Interface (ATAPI) bus.

VDSBusTypeAta: Disk resides on an AT Attached (ATA) bus.

VDSBusType1394: Disk resides on an IEEE 1394 bus.

VDSBusTypeSSA: Disk resides on a serial storage architecture (SSA) bus.

VSDBusTypeFibre: Disk resides on a **fiber channel bus**.

VDSBusTypeUsb: Disk resides on a **universal serial bus (USB)**.

VDSBusTypeRAID: Disk resides on a **RAID** bus.

VDSBusTypeiScsi: Disk resides on an **iSCSI** bus.

VDSBusTypeSas: Disk resides on a Serial Attached SCSI (SAS) bus.

VDSBusTypeSata: Disk resides on a Serial ATA (SATA) bus.

VDSBusTypeSd: Disk resides on an security descriptor (SD) bus.

VDSBusTypeMmc: Disk resides on a Microsoft Management Console (MMC) bus.

VDSBusTypeMax: Max bus type value. Note that this value does not identify a particular bus type, but rather, it serves as an end value of the enumeration.

VDSBusTypeMaxReserved: Max reserved bus type value. Bus type values below this range are reserved for use by Microsoft.

2.2.1.2.5 VDS_STORAGE_IDENTIFIER_CODE_SET

The **VDS_STORAGE_IDENTIFIER_CODE_SET** enumeration defines the code set that is used by the storage device identifier, as specified in [SPC-3] section 7.6.3.

```
typedef  enum _VDS_STORAGE_IDENTIFIER_CODE_SET
{
  VDSStorageIdCodeSetReserved = 0x00000000,
  VDSStorageIdCodeSetBinary = 0x00000001,
  VDSStorageIdCodeSetAscii = 0x00000002,
  VDSStorageIdCodeSetUtf8 = 0x00000003
} VDS_STORAGE_IDENTIFIER_CODE_SET;
```

VDSStorageIdCodeSetReserved: This value is reserved by the SPC-3 standard and is not used.

VDSStorageIdCodeSetBinary: The identifier contains binary values.

VDSStorageIdCodeSetAscii: The identifier contains ASCII values.

VDSStorageIdCodeSetUtf8: The identifier contains Universal Character Set Transformation Format 8 (UTF-8) values.

2.2.1.2.6 VDS_STORAGE_IDENTIFIER_TYPE

The **VDS_STORAGE_IDENTIFIER_TYPE** enumeration defines the types of storage device identifiers, as specified in [SPC-3] section 7.6.3.

```
typedef  enum _VDS_STORAGE_IDENTIFIER_TYPE
{
```

```
  VDSStorageIdTypeVendorSpecific = 0x00000000,
  VDSStorageIdTypeVendorId = 0x00000001,
  VDSStorageIdTypeEUI64 = 0x00000002,
  VDSStorageIdTypeFCPHName = 0x00000003,
  VDSStorageIdTypePortRelative = 0x00000004,
  VDSStorageIdTypeTargetPortGroup = 0x00000005,
  VDSStorageIdTypeLogicalUnitGroup = 0x00000006,
  VDSStorageIdTypeMD5LogicalUnitIdentifier = 0x00000007,
  VDSStorageIdTypeScsiNameString = 0x00000008
} VDS_STORAGE_IDENTIFIER_TYPE;
```

VDSStorageIdTypeVendorSpecific: Storage identifier is vendor-specific.

VDSStorageIdTypeVendorId: Storage identifier is a vendor identifier.

VDSStorageIdTypeEUI64: Storage identifier is a 64-bit **extended unique identifier (EUI-64)**.

VDSStorageIdTypeFCPHName: Storage identifier is a Fibre Channel Physical and Signaling Interface (FC-PH) identifier.

VDSStorageIdTypePortRelative: Storage identifier is a relative target port identifier.

VDSStorageIdTypeTargetPortGroup: Storage identifier is a target port group number.

VDSStorageIdTypeLogicalUnitGroup: Storage identifier is a logical unit group number.

VDSStorageIdTypeMD5LogicalUnitIdentifier: Storage identifier is an MD5 logical unit number (LUN).

VDSStorageIdTypeScsiNameString: Storage identifier is a **SCSI name string identifier**.

2.2.1.2.7 VDS_INTERCONNECT_ADDRESS_TYPE

The **VDS_INTERCONNECT_ADDRESS_TYPE** enumeration defines the set of valid address types of a physical interconnect.

```
typedef  enum _VDS_INTERCONNECT_ADDRESS_TYPE
{
  VDS_IA_UNKNOWN = 0x00000000,
  VDS_IA_FCFS = 0x00000001,
  VDS_IA_FCPH = 0x00000002,
  VDS_IA_FCPH3 = 0x00000003,
  VDS_IA_MAC = 0x00000004,
  VDS_IA_SCSI = 0x00000005
} VDS_INTERCONNECT_ADDRESS_TYPE;
```

VDS_IA_UNKNOWN: This value is reserved.

VDS_IA_FCFS: Address type is first come, first served.

VDS_IA_FCPH: Address type is FC-PH.

VDS_IA_FCPH3: Address type is FC-PH-3.

VDS_IA_MAC: Address type is media access control (MAC).

VDS_IA_SCSI: Address type is SCSI.

2.2.1.2.8 VDS_FILE_SYSTEM_TYPE

The **VDS_FILE_SYSTEM_TYPE** enumeration defines the set of valid types for a file system.

```
typedef  enum _VDS_FILE_SYSTEM_TYPE
{
  VDS_FST_UNKNOWN = 0x00000000,
  VDS_FST_RAW = 0x00000001,
  VDS_FST_FAT = 0x00000002,
  VDS_FST_FAT32 = 0x00000003,
  VDS_FST_NTFS = 0x00000004,
  VDS_FST_CDFS = 0x00000005,
  VDS_FST_UDF = 0x00000006
} VDS_FILE_SYSTEM_TYPE;
```

VDS_FST_UNKNOWN: The file system is unknown.

VDS_FST_RAW: The file system is raw.

VDS_FST_FAT: The file system is a **FAT file system**.

VDS_FST_FAT32: The file system is **FAT32**.

VDS_FST_NTFS: The file system is the NTFS file system.

VDS_FST_CDFS: The file system is the **compact disc file system (CDFS)**.

VDS_FST_UDF: The file system is **Universal Disk Format (UDF)**.

2.2.1.2.9 VDS_FILE_SYSTEM_FLAG

The **VDS_FILE_SYSTEM_FLAG** enumeration defines the set of valid flags for a file system format type.

If more than one flag is specified, the file system type supports all the file system allocation sizes that are specified. However, a specific file system on a volume does not have multiple allocation sizes at the same time.

```
typedef  enum _VDS_FILE_SYSTEM_FLAG
{
  VDS_FSF_SUPPORT_FORMAT = 0x00000001,
  VDS_FSF_SUPPORT_QUICK_FORMAT = 0x00000002,
  VDS_FSF_SUPPORT_COMPRESS = 0x00000004,
  VDS_FSF_SUPPORT_SPECIFY_LABEL = 0x00000008,
  VDS_FSF_SUPPORT_MOUNT_POINT = 0x00000010,
  VDS_FSF_SUPPORT_REMOVABLE_MEDIA = 0x00000020,
  VDS_FSF_SUPPORT_EXTEND = 0x00000040,
  VDS_FSF_ALLOCATION_UNIT_512 = 0x00010000,
  VDS_FSF_ALLOCATION_UNIT_1K = 0x00020000,
  VDS_FSF_ALLOCATION_UNIT_2K = 0x00040000,
  VDS_FSF_ALLOCATION_UNIT_4K = 0x00080000,
  VDS_FSF_ALLOCATION_UNIT_8K = 0x00100000,
```

```
  VDS_FSF_ALLOCATION_UNIT_16K = 0x00200000,
  VDS_FSF_ALLOCATION_UNIT_32K = 0x00400000,
  VDS_FSF_ALLOCATION_UNIT_64K = 0x00800000,
  VDS_FSF_ALLOCATION_UNIT_128K = 0x01000000,
  VDS_FSF_ALLOCATION_UNIT_256K = 0x02000000
} VDS_FILE_SYSTEM_FLAG;
```

VDS_FSF_SUPPORT_FORMAT: If set, the file system format type supports format.

VDS_FSF_SUPPORT_QUICK_FORMAT: If set, the file system format type supports quick format.

VDS_FSF_SUPPORT_COMPRESS: If set, the file system format type supports file compression.

VDS_FSF_SUPPORT_SPECIFY_LABEL: If set, the file system format type supports **file system labels**.

VDS_FSF_SUPPORT_MOUNT_POINT: If set, the file system format type supports mount points.

VDS_FSF_SUPPORT_REMOVABLE_MEDIA: If set, the file system format type supports **removable media**.

VDS_FSF_SUPPORT_EXTEND: If set, the file system format type supports extending volumes.

VDS_FSF_ALLOCATION_UNIT_512: If set, the file system format supports allocation units of 512 bytes.

VDS_FSF_ALLOCATION_UNIT_1K: If set, the file system format type supports allocation units of 1 kilobyte.

VDS_FSF_ALLOCATION_UNIT_2K: If set, the file system format type supports allocation units of 2 kilobytes.

VDS_FSF_ALLOCATION_UNIT_4K: If set, the file system format type supports allocation units of 4 kilobytes.

VDS_FSF_ALLOCATION_UNIT_8K: If set, the file system format type supports allocation units of 8 kilobytes.

VDS_FSF_ALLOCATION_UNIT_16K: If set, the file system format type supports allocation units of 16 kilobytes.

VDS_FSF_ALLOCATION_UNIT_32K: If set, the file system format type supports allocation units of 32 kilobytes.

VDS_FSF_ALLOCATION_UNIT_64K: If set, the file system format type supports allocation units of 64 kilobytes.

VDS_FSF_ALLOCATION_UNIT_128K: If set, the file system format type supports allocation units of 128 kilobytes.

VDS_FSF_ALLOCATION_UNIT_256K: If set, the file system format type supports allocation units of 256 kilobytes.

2.2.1.2.10 VDS_FILE_SYSTEM_PROP_FLAG

The **VDS_FILE_SYSTEM_PROP_FLAG** enumeration defines the set of fields for a file system. A value that accepts these flags MAY have the following flag set:

```
typedef  enum _VDS_FILE_SYSTEM_PROP_FLAG
{
  VDS_FPF_COMPRESSED = 0x00000001
} VDS_FILE_SYSTEM_PROP_FLAG;
```

VDS_FPF_COMPRESSED: If set, the file system supports file compression.

2.2.1.2.11 VDS_FILE_SYSTEM_FORMAT_SUPPORT_FLAG

The **VDS_FILE_SYSTEM_FORMAT_SUPPORT_FLAG** enumeration defines the properties of file systems that are supported for formatting volumes.<7>

```
typedef  enum _VDS_FILE_SYSTEM_FORMAT_SUPPORT_FLAG
{
  VDS_FSS_DEFAULT = 0x00000001,
  VDS_FSS_PREVIOUS_REVISION = 0x00000002,
  VDS_FSS_RECOMMENDED = 0x00000004
} VDS_FILE_SYSTEM_FORMAT_SUPPORT_FLAG;
```

VDS_FSS_DEFAULT: The file system is the default file system for formatting the volume.

VDS_FSS_PREVIOUS_REVISION: The revision of the file system is not the latest revision that is supported for formatting the volume.

VDS_FSS_RECOMMENDED: The file system is the recommended file system for formatting the volume.

2.2.1.2.12 VDS_DISK_EXTENT_TYPE

The **VDS_DISK_EXTENT_TYPE** enumeration defines the set of valid types for a disk extent.

```
typedef  enum _VDS_DISK_EXTENT_TYPE
{
  VDS_DET_UNKNOWN = 0x00000000,
  VDS_DET_FREE = 0x00000001,
  VDS_DET_DATA = 0x00000002,
  VDS_DET_OEM = 0x00000003,
  VDS_DET_ESP = 0x00000004,
  VDS_DET_MSR = 0x00000005,
  VDS_DET_LDM = 0x00000006,
  VDS_DET_UNUSABLE = 0x00007FFF
} VDS_DISK_EXTENT_TYPE;
```

VDS_DET_UNKNOWN: The extent belongs to an unknown partition type.

VDS_DET_FREE: The extent belongs to an area of **free space**.

VDS_DET_DATA: The extent belongs to a volume.

VDS_DET_OEM: The extent belongs to an OEM partition.

VDS_DET_ESP: The extent belongs to an **Extensible Firmware Interface (EFI) system partition**.

VDS_DET_MSR: The extent belongs to a Microsoft Reserved (MSR) partition.

VDS_DET_LDM: The extent belongs to a disk management metadata partition.

VDS_DET_UNUSABLE: The extent belongs to an area of unusable space.

2.2.1.2.13 VDS_PARTITION_STYLE

The **VDS_PARTITION_STYLE** enumeration defines the styles of partitions.

```
typedef  enum _VDS_PARTITION_STYLE
{
  VDS_PST_UNKNOWN = 0x00000000,
  VDS_PST_MBR = 0x00000001,
  VDS_PST_GPT = 0x00000002
} VDS_PARTITION_STYLE;
```

VDS_PST_UNKNOWN: The partition format is unknown.

VDS_PST_MBR: The partition format is **master boot record (MBR)**.

VDS_PST_GPT: The partition format is **GPT**.

2.2.1.2.14 VDS_PARTITION_FLAG

The **VDS_PARTITION_FLAG** enumeration defines flags that describe partitions. A value that accepts these flags MAY have the following flag set:

```
typedef  enum _VDS_PARTITION_FLAG
{
  VDS_PTF_SYSTEM = 0x00000001
} VDS_PARTITION_FLAG;
```

VDS_PTF_SYSTEM: Value that indicates that the partition is a system partition.

2.2.1.2.15 VDS_VOLUME_TYPE

The **VDS_VOLUME_TYPE** enumeration defines the set of valid types for a volume object.

```
typedef  enum _VDS_VOLUME_TYPE
{
  VDS_VT_UNKNOWN = 0x00000000,
  VDS_VT_SIMPLE = 0x0000000A,
  VDS_VT_SPAN = 0x0000000B,
  VDS_VT_STRIPE = 0x0000000C,
  VDS_VT_MIRROR = 0x0000000D,
  VDS_VT_PARITY = 0x0000000E
} VDS_VOLUME_TYPE;
```

VDS_VT_UNKNOWN: The status of the volume is unknown.

VDS_VT_SIMPLE: The volume type is simple: it is composed of extents from exactly one disk.

VDS_VT_SPAN: The volume type is spanned: it is composed of extents from more than one disk.

VDS_VT_STRIPE: The volume type is striped, which is equivalent to **RAID-0**.

VDS_VT_MIRROR: The volume type is mirrored, which is equivalent to **RAID-1**.

VDS_VT_PARITY: The volume type is striped with parity, which accounts for RAID levels 3, 4, 5, and 6.

2.2.1.3 Structures

2.2.1.3.1 VDS_PACK_NOTIFICATION

The **VDS_PACK_NOTIFICATION** structure provides information about a pack notification.

```
typedef struct _VDS_PACK_NOTIFICATION {
  unsigned long ulEvent;
  VDS_OBJECT_ID packId;
} VDS_PACK_NOTIFICATION;
```

ulEvent: The type of pack notification; it MUST be one of the following values:

Value	Meaning
VDS_NF_PACK_ARRIVE 0x00000001	The pack notification was newly created.
VDS_NF_PACK_DEPART 0x00000002	The pack notification was deleted.
VDS_NF_PACK_MODIFY 0x00000003	The pack notification was modified.

packId: The VDS object ID of the pack object to which the notification refers.

2.2.1.3.2 VDS_DISK_NOTIFICATION

The **VDS_DISK_NOTIFICATION** structure provides information about a disk notification.

```
typedef struct _VDS_DISK_NOTIFICATION {
  unsigned long ulEvent;
  VDS_OBJECT_ID diskId;
} VDS_DISK_NOTIFICATION;
```

ulEvent: The type of disk notification; it MUST be one of the following values:

Value	Meaning
VDS_NF_DISK_ARRIVE 0x00000008	The disk has become visible to the operating system.
VDS_NF_DISK_DEPART 0x00000009	The disk is no longer visible to the operating system.
VDS_NF_DISK_MODIFY 0x0000000A	The disk or its properties were modified.

diskId: The VDS object ID of the disk object to which the notification refers.

2.2.1.3.3 VDS_TRANSITION_STATE

The **VDS_TRANSITION_STATE** enumeration defines the set of valid transition state values for a VDS object.

```
typedef  enum _VDS_TRANSITION_STATE
{
  VDS_TS_UNKNOWN = 0x00000000,
  VDS_TS_STABLE = 0x00000001,
  VDS_TS_EXTENDING = 0x00000002,
  VDS_TS_SHRINKING = 0x00000003,
  VDS_TS_RECONFIGING = 0x00000004
} VDS_TRANSITION_STATE;
```

VDS_TS_UNKNOWN: The transition state of the object cannot be determined.

VDS_TS_STABLE: The object is stable. No configuration activity is currently in progress.

VDS_TS_EXTENDING: The object is being extended.

VDS_TS_SHRINKING: The object is being shrunk.

VDS_TS_RECONFIGING: The object is being automatically reconfigured.

2.2.1.3.4 VDS_VOLUME_NOTIFICATION

The **VDS_VOLUME_NOTIFICATION** structure provides information about a volume change notification.

```
typedef struct _VDS_VOLUME_NOTIFICATION {
  unsigned long ulEvent;
  VDS_OBJECT_ID volumeId;
  VDS_OBJECT_ID plexId;
  unsigned long ulPercentCompleted;
} VDS_VOLUME_NOTIFICATION;
```

ulEvent: Determines the volume event for which an application will be notified; it MUST be one of the following values:

Value	Meaning
VDS_NF_VOLUME_ARRIVE 0x00000004	A new volume is visible to the operating system.
VDS_NF_VOLUME_DEPART 0x00000005	An existing volume is no longer visible to the operating system.
VDS_NF_VOLUME_MODIFY 0x00000006	The volume was modified.
VDS_NF_VOLUME_REBUILDING_PROGRESS 0x00000007	A fault tolerant volume is being regenerated or resynchronized.

volumeId: The VDS object ID of the volume object to which the notification refers.

plexId: The VDS object ID of a volume plex object to which the notification refers, if any. VDS applies this identifier during the rebuild operation, which can execute on multiple plexes at different rates.

ulPercentCompleted: The percentage of completion for the operation. Valid values range from 0-100.

2.2.1.3.5 VDS_PARTITION_NOTIFICATION

The **VDS_PARTITION_NOTIFICATION** structure provides information about a partition notification.

```
typedef struct _VDS_PARTITION_NOTIFICATION {
  unsigned long ulEvent;
  VDS_OBJECT_ID diskId;
  ULONGLONG ullOffset;
} VDS_PARTITION_NOTIFICATION;
```

ulEvent: Determines the partition event for which an application will be notified; it MUST be one of the following values:

Value	Meaning
VDS_NF_PARTITION_ARRIVE 0x0000000B	A new partition is visible to the operating system.
VDS_NF_PARTITION_DEPART 0x0000000C	An existing partition is no longer visible to the operating system.
VDS_NF_PARTITION_MODIFY 0x0000000D	An existing partition changed.

diskId: The VDS object ID of the disk object containing the partition that triggered the event.

ullOffset: The byte offset of the partition from the beginning of the disk.

2.2.1.3.6 VDS_DRIVE_LETTER_NOTIFICATION

The **VDS_DRIVE_LETTER_NOTIFICATION** structure provides information about a drive letter notification.

```
typedef struct _VDS_DRIVE_LETTER_NOTIFICATION {
  unsigned long ulEvent;
  WCHAR wcLetter;
  VDS_OBJECT_ID volumeId;
} VDS_DRIVE_LETTER_NOTIFICATION;
```

ulEvent: Determines the drive-letter event for which an application will be notified; it MUST be one of the following values:

Value	Meaning
VDS_NF_DRIVE_LETTER_FREE 0x000000CC	The drive letter is no longer in use.
VDS_NF_DRIVE_LETTER_ASSIGN 0x000000CD	The drive letter has been assigned to a volume.

wcLetter: The drive letter that triggered the event, as a single uppercase or lowercase alphabetical (A-Z) Unicode character.

volumeId: The VDS object ID of the volume object to which the drive letter is assigned. If the drive letter is freed, the volume identifier is GUID_NULL.

2.2.1.3.7 VDS_FILE_SYSTEM_NOTIFICATION

The **VDS_FILE_SYSTEM_NOTIFICATION** structure provides information about a file system notification.

```
typedef struct _VDS_FILE_SYSTEM_NOTIFICATION {
  unsigned long ulEvent;
  VDS_OBJECT_ID volumeId;
  DWORD dwPercentCompleted;
} VDS_FILE_SYSTEM_NOTIFICATION;
```

ulEvent: Determines the file system event for which an application will be notified; it MUST be one of the following values:

Value	Meaning
VDS_NF_FILE_SYSTEM_MODIFY 0x000000CE	A volume received a new label, or a file system was extended or shrunk; does not include a change to the file system compression flags.
VDS_NF_FILE_SYSTEM_FORMAT_PROGRESS 0x000000CF	A file system is being formatted.

Value	Meaning
VDS_NF_FILE_SYSTEM_SHRINKING_PROGRESS 0x000000D1	A volume is running compression operations.

volumeId: The VDS object ID of the volume object containing the file system that triggered the event.

dwPercentCompleted: The completed format progress as a percentage of the whole.

2.2.1.3.8 VDS_MOUNT_POINT_NOTIFICATION

The **VDS_MOUNT_POINT_NOTIFICATION** structure provides information about a mount point change notification.

```
typedef struct _VDS_MOUNT_POINT_NOTIFICATION {
  unsigned long ulEvent;
  VDS_OBJECT_ID volumeId;
} VDS_MOUNT_POINT_NOTIFICATION;
```

ulEvent: Determines the mount point event for which an application will be notified; it MUST be the following value:

Value	Meaning
VDS_NF_MOUNT_POINTS_CHANGE 0x000000D0	The mount point changed.

volumeId: The VDS object ID of the volume object containing the mount point that triggered the event.

2.2.1.3.9 VDS_NOTIFICATION

The **VDS_NOTIFICATION** structure provides information about a notification.

```
typedef struct _VDS_NOTIFICATION {
  VDS_NOTIFICATION_TARGET_TYPE objectType;
  [switch_is(objectType)] union {
    [case(VDS_NTT_PACK)]
      VDS_PACK_NOTIFICATION Pack;
    [case(VDS_NTT_DISK)]
      VDS_DISK_NOTIFICATION Disk;
    [case(VDS_NTT_VOLUME)]
      VDS_VOLUME_NOTIFICATION Volume;
    [case(VDS_NTT_PARTITION)]
      VDS_PARTITION_NOTIFICATION Partition;
    [case(VDS_NTT_DRIVE_LETTER)]
      VDS_DRIVE_LETTER_NOTIFICATION Letter;
    [case(VDS_NTT_FILE_SYSTEM)]
      VDS_FILE_SYSTEM_NOTIFICATION FileSystem;
    [case(VDS_NTT_MOUNT_POINT)]
      VDS_MOUNT_POINT_NOTIFICATION MountPoint;
    [default]
```

```
      ;
  };
} VDS_NOTIFICATION;
```

objectType: A value defined in the **VDS_NOTIFICATION_TARGET_TYPE** enumeration that describes the type of notification.

Pack: A **VDS_PACK_NOTIFICATION** structure that describes a pack change.

Disk: A **VDS_DISK_NOTIFICATION** structure that describes a disk change.

Volume: A **VDS_VOLUME_NOTIFICATION** structure that describes a volume change.

Partition: A **VDS_PARTITION_NOTIFICATION** structure that describes a partition change.

Letter: A **VDS_DRIVE_LETTER_NOTIFICATION** structure that describes a drive letter change.

FileSystem: A **VDS_FILE_SYSTEM_NOTIFICATION** structure that describes a file system change.

MountPoint: A **VDS_MOUNT_POINT_NOTIFICATION** structure that describes a mount point change.

2.2.1.3.10 VDS_ASYNC_OUTPUT

The **VDS_ASYNC_OUTPUT** structure provides information from a completed asynchronous operation.

```
typedef struct _VDS_ASYNC_OUTPUT {
  VDS_ASYNC_OUTPUT_TYPE type;
  [switch_is(type)] union {
    [case(VDS_ASYNCOUT_CREATEPARTITION)]
      struct _cp {
      ULONGLONG ullOffset;
      VDS_OBJECT_ID volumeId;
    } cp;
    [case(VDS_ASYNCOUT_CREATEVOLUME)]
      struct _cv {
      IUnknown* pVolumeUnk;
    } cv;
    [case(VDS_ASYNCOUT_BREAKVOLUMEPLEX)]
      struct _bvp {
      IUnknown* pVolumeUnk;
    } bvp;
    [case(VDS_ASYNCOUT_SHRINKVOLUME)]
      struct _sv {
      ULONGLONG ullReclaimedBytes;
    } sv;
    [default]
      ;
  };
} VDS_ASYNC_OUTPUT;
```

type: A value from the **VDS_ASYNC_OUTPUT_TYPE** enumeration that indicates the type of operation information.

cp: The cp structure provides information about a newly created partition.

ullOffset: The byte offset of the partition from the beginning of the disk.

volumeId: The VDS object ID of the associated volume object, if the partition is a volume.

cv: The cv structure provides information about a newly created volume.

pVolumeUnk: A pointer to the **IUnknown** interface of the newly created volume.

bvp: The bvp structure provides information about a volume after a plex is broken.

pVolumeUnk: A pointer to the **IUnknown** interface of the volume that was broken off.

sv: The sv structure provides information about a volume shrink operation.

ullReclaimedBytes: The number of bytes that the volume shrink operation reclaimed.

2.2.1.3.11 VDS_PARTITION_INFO_MBR

The **VDS_PARTITION_INFO_MBR** structure provides information about an MBR partition.

```
typedef struct {
  byte partitionType;
  boolean bootIndicator;
  boolean recognizedPartition;
  DWORD hiddenSectors;
} VDS_PARTITION_INFO_MBR;
```

partitionType: The byte value indicating the partition type.<8>

bootIndicator: A Boolean value that indicates whether the partition is bootable.

recognizedPartition: A Boolean value that indicates whether the partition type is supported.

hiddenSectors: The number of **sectors** between the start of the partition and the partition's first usable area.

2.2.1.3.12 VDS_PARTITION_INFO_GPT

The **VDS_PARTITION_INFO_GPT** structure provides information about a partition in a GPT.

```
typedef struct _VDS_PARTITION_INFO_GPT {
  GUID partitionType;
  GUID partitionId;
  ULONGLONG attributes;
  WCHAR name[24];
} VDS_PARTITION_INFO_GPT;
```

partitionType: A GUID indicating the partition type.<9>

partitionId: The GUID of the partition.

attributes: The attributes of the partition; they may have a combination of the following values:

Value	Meaning
GPT_ATTRIBUTE_PLATFORM_REQUIRED 0x0000000000000001	Partition is required for the platform to function properly.<10>
GPT_BASIC_DATA_ATTRIBUTE_READ_ONLY 0x1000000000000000	Partition cannot be written to, but can be read from. Used only with the basic data partition type.
GPT_BASIC_DATA_ATTRIBUTE_SHADOW_COPY 0x2000000000000000	Partition is a shadow copy. Used only with the basic data partition type.
GPT_BASIC_DATA_ATTRIBUTE_HIDDEN 0x4000000000000000	Partition is hidden and will not be mounted. Used only with the basic data partition type.
GPT_BASIC_DATA_ATTRIBUTE_NO_DRIVE_LETTER 0x8000000000000000	Partition does not receive a drive letter by default when moving the disk to another machine. Used only with the basic data partition type.

name: Null-terminated Unicode name of the partition.

2.2.1.3.13 VDS_STORAGE_IDENTIFIER

The **VDS_STORAGE_IDENTIFIER** structure provides information about a storage identifier.

```
typedef struct _VDS_STORAGE_IDENTIFIER {
  VDS_STORAGE_IDENTIFIER_CODE_SET m_CodeSet;
  VDS_STORAGE_IDENTIFIER_TYPE m_Type;
  unsigned long m_cbIdentifier;
  [size_is(m_cbIdentifier)] byte* m_rgbIdentifier;
} VDS_STORAGE_IDENTIFIER;
```

m_CodeSet: Value from the **VDS_STORAGE_IDENTIFIER_CODE_SET** enumeration that defines the code set of the storage identifier.

m_Type: Value from the **VDS_STORAGE_IDENTIFIER_TYPE** enumeration that defines the type of the storage identifier.

m_cbIdentifier: Length of the m_rgbIdentifier identifier in bytes.

m_rgbIdentifier: Value of the storage identifier. These identifiers depend on both the code set and the type.

2.2.1.3.14 VDS_STORAGE_DEVICE_ID_DESCRIPTOR

The **VDS_STORAGE_DEVICE_ID_DESCRIPTOR** structure provides information about a device identification descriptor.

```
typedef struct _VDS_STORAGE_DEVICE_ID_DESCRIPTOR {
  unsigned long m_version;
  unsigned long m_cIdentifiers;
  [size_is(m_cIdentifiers)] VDS_STORAGE_IDENTIFIER* m_rgIdentifiers;
} VDS_STORAGE_DEVICE_ID_DESCRIPTOR;
```

m_version: The version number of the **VDS_STORAGE_DEVICE_ID_DESCRIPTOR** structure as specified by the device manufacturer and in [SPC-3].

m_cIdentifiers: The number of elements in the m_rgIdentifiers array.

m_rgIdentifiers: The array of **VDS_STORAGE_IDENTIFIER** structures that contain the storage identifier information.

2.2.1.3.15 VDS_INTERCONNECT

The **VDS_INTERCONNECT** structure defines the address data of a physical interconnect, as specified in [SPC-3].

```
typedef struct _VDS_INTERCONNECT {
  VDS_INTERCONNECT_ADDRESS_TYPE m_addressType;
  unsigned long m_cbPort;
  [size_is(m_cbPort)] byte* m_pbPort;
  unsigned long m_cbAddress;
  [size_is(m_cbAddress)] byte* m_pbAddress;
} VDS_INTERCONNECT;
```

m_addressType: A **VDS_INTERCONNECT_ADDRESS_TYPE** structure that stores the address type of the interconnect.

m_cbPort: The size, in bytes, of the interconnect address data for the LUN port to which **m_pbPort** refers.

m_pbPort: A pointer to the interconnect address data for the LUN port.

m_cbAddress: The size, in bytes, of the interconnect address data for the LUN to which **m_pbAddress** refers.

m_pbAddress: A pointer to the interconnect address data for the LUN.

2.2.1.3.16 VDS_LUN_INFORMATION

The **VDS_LUN_INFORMATION** structure provides information about a SCSI-2 device.

```
typedef struct _VDS_LUN_INFORMATION {
  unsigned long m_version;
  byte m_DeviceType;
```

```
  byte m_DeviceTypeModifier;
  long m_bCommandQueueing;
  VDS_STORAGE_BUS_TYPE m_BusType;
  [string] char* m_szVendorId;
  [string] char* m_szProductId;
  [string] char* m_szProductRevision;
  [string] char* m_szSerialNumber;
  GUID m_diskSignature;
  VDS_STORAGE_DEVICE_ID_DESCRIPTOR m_deviceIdDescriptor;
  unsigned long m_cInterconnects;
  [size_is(m_cInterconnects)] VDS_INTERCONNECT* m_rgInterconnects;
} VDS_LUN_INFORMATION;
```

m_version: The version number of the **VDS_LUN_INFORMATION** structure. As of the current version of this protocol, this value is always VER_VDS_LUN_INFORMATION (0x00000001).

m_DeviceType: The SCSI-2 device type of the device, as specified in [SPC-3].

m_DeviceTypeModifier: The SCSI-2 device type modifier, if any, as specified in [SPC-3].

m_bCommandQueueing: A Boolean value that indicates whether the device supports multiple outstanding commands.

m_BusType: A value from the **VDS_STORAGE_BUS_TYPE** enumeration that indicates the bus type of the device.

m_szVendorId: The null-terminated vendor identification Unicode string of the device. This value is NULL if no vendor ID exists.

m_szProductId: The null-terminated product identification Unicode string of the device. This value is NULL if no product ID exists.

m_szProductRevision: The null-terminated product revision Unicode string of the device. This value is NULL if no product revision information exists.

m_szSerialNumber: The null-terminated serial number of the device. This value is NULL if no serial number exists.

m_diskSignature: The **disk signature** of the disk.

m_deviceIdDescriptor: A **VDS_STORAGE_DEVICE_ID_DESCRIPTOR** structure that contains the identification descriptor of the device.

m_cInterconnects: The number of elements in the m_rgInterconnects array.

m_rgInterconnects: Any array of **VDS_INTERCONNECT** structures that describe the physical interconnects to the device.

2.2.1.3.17 VDS_FILE_SYSTEM_PROP

The **VDS_FILE_SYSTEM_PROP** structure provides information about the properties of a file system.

```
typedef struct _VDS_FILE_SYSTEM_PROP {
  VDS_FILE_SYSTEM_TYPE type;
```

```
  VDS_OBJECT_ID volumeId;
  unsigned long ulFlags;
  ULONGLONG ullTotalAllocationUnits;
  ULONGLONG ullAvailableAllocationUnits;
  unsigned long ulAllocationUnitSize;
  [string] WCHAR* pwszLabel;
} VDS_FILE_SYSTEM_PROP,
 *PVDS_FILE_SYSTEM_PROP;
```

type: A **VDS_FILE_SYSTEM_TYPE** structure that provides information about the type of the file system.

volumeId: The VDS object ID of the volume object on which the file system resides.

ulFlags: The combination of any values, by using the bitwise OR operator, that are defined in the **VDS_FILE_SYSTEM_PROP_FLAG** enumeration.

ullTotalAllocationUnits: The total number of allocation units on the file system.

ullAvailableAllocationUnits: The number of allocation units available on the file system.

ulAllocationUnitSize: The size of the **allocation units** in use by the file system.

pwszLabel: A null-terminated Unicode label of the file system.

2.2.1.3.18 VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP

The **VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP** structure provides information about file systems that are supported for formatting volumes.

```
typedef struct _VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP {
  unsigned long ulFlags;
  unsigned short usRevision;
  unsigned long ulDefaultUnitAllocationSize;
  unsigned long rgulAllowedUnitAllocationSizes[20];
  WCHAR wszName[20];
} VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP,
 *PVDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP;
```

ulFlags: The combination of any values, by using the bitwise OR operator, that are defined in the **VDS_FILE_SYSTEM_FORMAT_SUPPORT_FLAG** enumeration.

usRevision: A 16-bit, binary-coded decimal number that indicates the file system version, if any. The first two (most significant) digits (8-bits) indicate the major version while the last two (least significant) digits (8-bits) indicate the minor version. For example, a value that has a bit pattern of 00000010 01010000 (0x0250 in hexadecimal) represents version 2.50; 0x1195 represents version 11.95, and so on.

ulDefaultUnitAllocationSize: The default allocation unit size, in bytes, that the file system uses for formatting the volume. This value MUST be a power of 2 and MUST also appear in **rgulAllowedUnitAllocationSizes**.

rgulAllowedUnitAllocationSizes: A zero-terminated array of allocation unit sizes, in bytes, that the file system supports for formatting the volume. An array is not zero-terminated if the array contains 20 elements. Each of the values in the array MUST be a power of 2.

wszName: A null-terminated Unicode wide-character string that indicates the name of the file system.<11>

2.2.1.3.19 VDS_DISK_EXTENT

The **VDS_DISK_EXTENT** structure provides information about a disk extent.

```
typedef struct _VDS_DISK_EXTENT {
  VDS_OBJECT_ID diskId;
  VDS_DISK_EXTENT_TYPE type;
  ULONGLONG ullOffset;
  ULONGLONG ullSize;
  VDS_OBJECT_ID volumeId;
  VDS_OBJECT_ID plexId;
  unsigned long memberIdx;
} VDS_DISK_EXTENT,
 *PVDS_DISK_EXTENT;
```

diskId: The VDS object ID of the disk object on which the extent resides.

type: The value from the **VDS_DISK_EXTENT_TYPE** enumeration that indicates the type of the extent.

ullOffset: The byte offset of the disk extent from the beginning of the disk.

ullSize: The size, in bytes, of the extent.

volumeId: The VDS object ID of the volume object to which the extent belongs, if any.

plexId: The VDS object ID of the volume plex object to which the extent belongs, if it belongs to a volume.

memberIdx: The zero-based index of the **volume plex member** to which the extent belongs, if it belongs to a volume plex.

2.2.1.3.20 VDS_PARTITION_PROP

The **VDS_PARTITION_PROP** structure provides information about partition properties.

```
typedef struct _VDS_PARTITION_PROP {
  VDS_PARTITION_STYLE PartitionStyle;
  unsigned long ulFlags;
  unsigned long ulPartitionNumber;
  ULONGLONG ullOffset;
  ULONGLONG ullSize;
  [switch_is(PartitionStyle)] union {
    [case(VDS_PST_MBR)]
      VDS_PARTITION_INFO_MBR Mbr;
    [case(VDS_PST_GPT)]
      VDS_PARTITION_INFO_GPT Gpt;
    [default]
```

```
        ;
    };
} VDS_PARTITION_PROP;
```

PartitionStyle: The value from the **VDS_PARTITION_STYLE** enumeration that describes the partition format of the disk where the partition resides.

ulFlags: The combination of any values, by using the bitwise OR operator, from the **VDS_PARTITION_FLAG** enumeration describing the partition.

ulPartitionNumber: The one-based index number of the partition that the operating system assigns.

ullOffset: The byte offset of the partition from the beginning of the disk.

ullSize: The size of the partition, in bytes.

Mbr: A **VDS_PARTITION_INFO_MBR** structure that describes the MBR partition.

Gpt: A **VDS_PARTITION_INFO_GPT** structure that describes the GPT partition.

2.2.1.3.21 VDS_INPUT_DISK

The **VDS_INPUT_DISK** structure provides information about a disk for volume creation and modification.

```
typedef struct _VDS_INPUT_DISK {
  VDS_OBJECT_ID diskId;
  ULONGLONG ullSize;
  VDS_OBJECT_ID plexId;
  unsigned long memberIdx;
} VDS_INPUT_DISK;
```

diskId: The VDS object ID of the disk object.

ullSize: The size of the disk to use, in bytes.

plexId: When extending a volume, the VDS object ID of the plex object to which the disk will be added. A volume can only be extended by extending all members of all plexes in the same operation. This member is used when extending any volume and ignored when creating a volume or repairing a RAID-5 volume.

memberIdx: The zero-based member index of the disk to which the extent belongs. Either specify a memberIdx for all disks or specify it for none. VDS uses disks with the same memberIdx in the order they appear in the array. For example, the first disk in the array is always used first, even if it does not have the lowest index. This member is ignored when repairing a RAID 5 volume.

2.2.1.3.22 CREATE_PARTITION_PARAMETERS

The **VDS_PARTITION_PROP** structure provides information about partition properties.

```
typedef struct _CREATE_PARTITION_PARAMETERS {
  VDS_PARTITION_STYLE style;
  [switch_is(style)] union {
    [case(VDS_PST_MBR)]
      struct {
      byte partitionType;
      boolean bootIndicator;
    } MbrPartInfo;
    [case(VDS_PST_GPT)]
      struct {
      GUID partitionType;
      GUID partitionId;
      ULONGLONG attributes;
      WCHAR name[24];
    } GptPartInfo;
  };
} CREATE_PARTITION_PARAMETERS;
```

style: A value from the **VDS_PARTITION_STYLE** enumeration that describes the disk partition format.

MbrPartInfo: Contains information for an MBR partition.

partitionType: The byte value that indicates the partition type to create.

bootIndicator: A Boolean value that indicates whether the partition is bootable.

GptPartInfo: Contains information for a GPT partition.

partitionType: A GUID that indicates the partition type to create.<12>

partitionId: The GUID of the partition.

attributes: A bitwise OR operator of attributes that is used to create the partition; it can have a combination of the following values:

Value	Meaning
GPT_ATTRIBUTE_PLATFORM_REQUIRED 0x0000000000000001	A partition is required for the platform to function properly.<13>
GPT_BASIC_DATA_ATTRIBUTE_READ_ONLY 0x1000000000000000	The partition can be read from, but not written to. Used only with the basic data partition type.
GPT_BASIC_DATA_ATTRIBUTE_HIDDEN 0x4000000000000000	The partition is hidden and is not mounted. Used only with the basic data partition type.
GPT_BASIC_DATA_ATTRIBUTE_NO_DRIVE_LETTER 0x8000000000000000	The partition does not receive a drive letter by default when moving the disk to another computer. Used only with the basic data partition type.

name: The null-terminated Unicode name of the partition.

2.2.2 Interface-Specific Data Types

2.2.2.1 IVdsService Data Types

This section lists data types that are used exclusively by methods in the **IVdsService** interface .

2.2.2.1.1 Data Types

2.2.2.1.1.1 MAX_FS_NAME_SIZE

Constant/value	Description
MAX_FS_NAME_SIZE 0x00000008	The **MAX_FS_NAME_SIZE** defines the maximum character length of a file system name.

2.2.2.1.2 Enumerations

2.2.2.1.2.1 VDS_OBJECT_TYPE

The **VDS_OBJECT_TYPE** enumeration defines the set of valid VDS object types.

```
typedef  enum _VDS_OBJECT_TYPE
{
  VDS_OT_UNKNOWN = 0x00000000,
  VDS_OT_PROVIDER = 0x00000001,
  VDS_OT_PACK = 0x0000000A,
  VDS_OT_VOLUME = 0x0000000B,
  VDS_OT_VOLUME_PLEX = 0x0000000C,
  VDS_OT_DISK = 0x0000000D,
  VDS_OT_HBAPORT = 0x0000005A,
  VDS_OT_INIT_ADAPTER = 0x0000005B,
  VDS_OT_INIT_PORTAL = 0x0000005C,
  VDS_OT_ASYNC = 0x00000064,
  VDS_OT_ENUM = 0x00000065
} VDS_OBJECT_TYPE;
```

VDS_OT_UNKNOWN: The object has an unknown type.

VDS_OT_PROVIDER: The object is a provider.

VDS_OT_PACK: The object is a pack (a **disk group**).

VDS_OT_VOLUME: The object is a volume.

VDS_OT_VOLUME_PLEX: The object is a plex of a volume.

VDS_OT_DISK: The object is a disk.

VDS_OT_HBAPORT: The object is an HBA port.

VDS_OT_INIT_ADAPTER: The object is an iSCSI initiator adapter.

VDS_OT_INIT_PORTAL: The object is an iSCSI initiator portal.

VDS_OT_ASYNC: The object maintains the status of an asynchronous VDS operation.

VDS_OT_ENUM: The object is an enumerator that contains an enumeration of other VDS objects.

2.2.2.1.2.2 VDS_SERVICE_FLAG

The **VDS_SERVICE_FLAG** enumeration defines the properties of the service.

```
typedef  enum _VDS_SERVICE_FLAG
{
  VDS_SVF_SUPPORT_DYNAMIC = 0x00000001,
  VDS_SVF_SUPPORT_FAULT_TOLERANT = 0x00000002,
  VDS_SVF_SUPPORT_GPT = 0x00000004,
  VDS_SVF_SUPPORT_DYNAMIC_1394 = 0x00000008,
  VDS_SVF_CLUSTER_SERVICE_CONFIGURED = 0x00000010,
  VDS_SVF_AUTO_MOUNT_OFF = 0x00000020,
  VDS_SVF_OS_UNINSTALL_VALID = 0x00000040,
  VDS_SVF_EFI = 0x00000080
} VDS_SERVICE_FLAG;
```

VDS_SVF_SUPPORT_DYNAMIC: The server supports **dynamic disks**.

VDS_SVF_SUPPORT_FAULT_TOLERANT: The server supports **fault-tolerant** disks.

VDS_SVF_SUPPORT_GPT: The server supports the GPT partition format.

VDS_SVF_SUPPORT_DYNAMIC_1394: The server supports dynamic IEEE 1394 disks.

VDS_SVF_CLUSTER_SERVICE_CONFIGURED: The server is running on a **cluster**.

VDS_SVF_AUTO_MOUNT_OFF: The server will not automatically mount disks.

VDS_SVF_OS_UNINSTALL_VALID: The server has an uninstall image to which it can roll back.

VDS_SVF_EFI: The computer starts an EFI from a GPT partition.

2.2.2.1.2.3 VDS_QUERY_PROVIDER_FLAG

The **VDS_QUERY_PROVIDER_FLAG** enumeration defines the set of valid flags for provider query operations. Callers can query for hardware providers, software providers, or both.<14>

```
typedef  enum _VDS_QUERY_PROVIDER_FLAG
{
  VDS_QUERY_SOFTWARE_PROVIDERS = 0x00000001,
  VDS_QUERY_HARDWARE_PROVIDERS = 0x00000002
} VDS_QUERY_PROVIDER_FLAG;
```

VDS_QUERY_SOFTWARE_PROVIDERS: If set, the operation queries for software providers.

VDS_QUERY_HARDWARE_PROVIDERS: If set, the operation queries for hardware providers.

2.2.2.1.2.4 VDS_DRIVE_LETTER_FLAG

The **VDS_DRIVE_LETTER_FLAG** enumeration defines the set of valid flags for a drive letter.

```
typedef  enum _VDS_DRIVE_LETTER_FLAG
{
  VDS_DLF_NON_PERSISTENT = 0x00000001
} VDS_DRIVE_LETTER_FLAG;
```

VDS_DLF_NON_PERSISTENT: If set, the drive letter no long appears after the computer is restarted.

2.2.2.1.3 Structures

2.2.2.1.3.1 VDS_SERVICE_PROP

The **VDS_SERVICE_PROP** structure provides information about the properties of a service.

```
typedef struct _VDS_SERVICE_PROP {
  [string] WCHAR* pwszVersion;
  unsigned long ulFlags;
} VDS_SERVICE_PROP;
```

pwszVersion: The version of VDS; a human-readable, null-terminated Unicode string. This string can be any human-readable, null-terminated Unicode value.<15>

ulFlags: A combination of any values, by using the bitwise OR operator, that is defined in the **VDS_SERVICE_FLAG** enumeration.

2.2.2.1.3.2 VDS_DRIVE_LETTER_PROP

The **VDS_DRIVE_LETTER_PROP** structure provides information about a drive letter.

```
typedef struct _VDS_DRIVE_LETTER_PROP {
  WCHAR wcLetter;
  VDS_OBJECT_ID volumeId;
  unsigned long ulFlags;
  long bUsed;
} VDS_DRIVE_LETTER_PROP,
 *PVDS_DRIVE_LETTER_PROP;
```

wcLetter: The drive letter as a single uppercase or lowercase alphabetical (A-Z) Unicode character.

volumeId: The VDS object ID of the volume object to which the drive letter is assigned. If the drive letter is not assigned to any volume, the value MUST be GUID_NULL.

ulFlags: The combination of any values, by using a bitwise OR operator, that is defined in the **VDS_DRIVE_LETTER_FLAG** enumeration.

bUsed: A Boolean value that indicates whether the drive letter is already in use.

2.2.2.1.3.3 VDS_FILE_SYSTEM_TYPE_PROP

The **VDS_FILE_SYSTEM_TYPE_PROP** structure provides information about a file system format.

```
typedef struct _VDS_FILE_SYSTEM_TYPE_PROP {
  VDS_FILE_SYSTEM_TYPE type;
  WCHAR wszName[8];
  unsigned long ulFlags;
  unsigned long ulCompressionFlags;
  unsigned long ulMaxLabelLength;
  [string] WCHAR* pwszIllegalLabelCharSet;
} VDS_FILE_SYSTEM_TYPE_PROP,
 *PVDS_FILE_SYSTEM_TYPE_PROP;
```

type: A value from the **VDS_FILE_SYSTEM_TYPE** enumeration that indicates the file system format type.

wszName: A null-terminated Unicode name of the file system format, for example, **NTFS** or FAT32.

ulFlags: A combination of any values, by using a bitwise OR operator, that are defined in the **VDS_FILE_SYSTEM_FLAG** enumeration.

ulCompressionFlags: A bitwise OR operator of any allocation units that are defined in the **VDS_FILE_SYSTEM_PROP_FLAG** enumeration.

ulMaxLabelLength: The maximum allowable length of a label for the file system format.

pwszIllegalLabelCharSet: A null-terminated sequence of Unicode characters that are not allowed in the label of the file system format.

2.2.2.2 IVdsServiceIscsi Data Types

This section lists data types that the **IVdsServiceIscsi** methods of the **IVdsServiceIscsi** interface use exclusively.

2.2.2.2.1 Structures

2.2.2.2.1.1 VDS_ISCSI_SHARED_SECRET

The **VDS_ISCSI_SHARED_SECRET** structure defines the **Challenge Handshake Authentication Protocol (CHAP)**, as specified in MS-CHAP], **shared secret** for an iSCSI initiator.

```
typedef struct _VDS_ISCSI_SHARED_SECRET {
  [size_is(ulSharedSecretSize)] unsigned char* pSharedSecret;
  unsigned long ulSharedSecretSize;
} VDS_ISCSI_SHARED_SECRET;
```

pSharedSecret: A pointer to an array of bytes that contains the secret.

ulSharedSecretSize: The number of bytes contained in the array that **pSharedSecret** references. Bytes MUST be at least 12, and less than or equal to 16.<16>

2.2.2.3 IVdsHbaPort Data Types

This section lists data types that the IVdsHbaPort methods of the **IVdsHbaPort** interface use exclusively.

2.2.2.3.1 Enumerations

2.2.2.3.1.1 VDS_HBAPORT_TYPE

The **VDS_HBAPORT_TYPE** enumeration defines the set of valid types for an HBA port. These types correspond to the HBA_PORTTYPE values, as specified in [HBAAPI]. These values are used in the type member of the **VDS_HBAPORT_PROP** structure.

```
typedef  enum _VDS_HBAPORT_TYPE
{
  VDS_HPT_UNKNOWN = 0x00000001,
  VDS_HPT_OTHER = 0x00000002,
  VDS_HPT_NOTPRESENT = 0x00000003,
  VDS_HPT_NPORT = 0x00000005,
  VDS_HPT_NLPORT = 0x00000006,
  VDS_HPT_FLPORT = 0x00000007,
  VDS_HPT_FPORT = 0x00000008,
  VDS_HPT_EPORT = 0x00000009,
  VDS_HPT_GPORT = 0x0000000A,
  VDS_HPT_LPORT = 0x00000014,
  VDS_HPT_PTP = 0x00000015
} VDS_HBAPORT_TYPE;
```

VDS_HPT_UNKNOWN: The port type is unknown.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_UNKNOWN

VDS_HPT_OTHER: The port type is another (undefined) type.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_OTHER

VDS_HPT_NOTPRESENT: The port type is not present.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_NOTPRESENT

VDS_HPT_NPORT: The port type is a fabric.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_NPORT

VDS_HPT_NLPORT: The port type is a public loop.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_NLPORT

VDS_HPT_FLPORT: The port type is a fabric on a loop.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_FLPORT

VDS_HPT_FPORT: The port type is a fabric port.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_FPORT

VDS_HPT_EPORT: The port type is a fabric expansion port.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_EPORT

VDS_HPT_GPORT: The port type is a generic fabric port.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_GPORT

VDS_HPT_LPORT: The port type is a private loop.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_LPORT

VDS_HPT_PTP: The port type is point-to-point.

Corresponding HBA_PORTTYPE value: HBA_PORTTYPE_PTP

2.2.2.3.1.2 VDS_HBAPORT_STATUS

The **VDS_HBAPORT_STATUS** enumeration defines the set of valid statuses for an HBA port. These values are used in the status member of the **VDS_HBAPORT_PROP** structure. These states correspond to the HBA_PORTSTATE values, as specified in [HBAAPI].<17>

```
typedef  enum _VDS_HBAPORT_STATUS
{
  VDS_HPS_UNKNOWN = 0x00000001,
  VDS_HPS_ONLINE = 0x00000002,
  VDS_HPS_OFFLINE = 0x00000003,
  VDS_HPS_BYPASSED = 0x00000004,
  VDS_HPS_DIAGNOSTICS = 0x00000005,
  VDS_HPS_LINKDOWN = 0x00000006,
  VDS_HPS_ERROR = 0x00000007,
  VDS_HPS_LOOPBACK = 0x00000008
} VDS_HBAPORT_STATUS;
```

VDS_HPS_UNKNOWN: The HBA port status is unknown.

Corresponding HBA_PORTSTATE value: HBA_PORTSTATE_UNKNOWN

VDS_HPS_ONLINE: The HBA port is operational.

Corresponding HBA_PORTSTATE value: HBA_PORTSTATE_ONLINE

VDS_HPS_OFFLINE: The HBA port was set **offline** by a user.

Corresponding HBA_PORTSTATE value: HBA_PORTSTATE_OFFLINE

VDS_HPS_BYPASSED: The HBA port is bypassed.

Corresponding HBA_PORTSTATE value: HBA_PORTSTATE_BYPASSED

VDS_HPS_DIAGNOSTICS: The HBA port is in diagnostics mode.

Corresponding HBA_PORTSTATE value: HBA_PORTSTATE_DIAGNOSTICS

VDS_HPS_LINKDOWN: The HBA port link is down.

Corresponding HBA_PORTSTATE value: HBA_PORTSTATE_LINKDOWN

VDS_HPS_ERROR: The HBA port has an error.

Corresponding HBA_PORTSTATE value: HBA_PORTSTATE_ERROR

VDS_HPS_LOOPBACK: The HBA port is loopback.

Corresponding HBA_PORTSTATE value: HBA_PORTSTATE_LOOPBACK

2.2.2.3.1.3 VDS_HBAPORT_SPEED_FLAG

The **VDS_HBAPORT_SPEED_FLAG** enumeration type defines the set of valid flags for determining the speeds that an HBA port supports. These values are used in the ulPortSpeed member of the VDS_HBAPORT_PROP structure. These flags correspond to the HBA_PORTSPEED flags, as specified in [HBAAPI].<18>

```
typedef  enum _VDS_HBAPORT_SPEED_FLAG
{
  VDS_HSF_UNKNOWN = 0x00000000,
  VDS_HSF_1GBIT = 0x00000001,
  VDS_HSF_2GBIT = 0x00000002,
  VDS_HSF_10GBIT = 0x00000004,
  VDS_HSF_4GBIT = 0x00000008,
  VDS_HSF_NOT_NEGOTIATED = 0x00008000
} VDS_HBAPORT_SPEED_FLAG;
```

VDS_HSF_UNKNOWN: The HBA port speed is unknown.

Corresponding HBA_PORTSPEED value: HBA_PORTSPEED_UNKNOWN

VDS_HSF_1GBIT: The HBA port supports a transfer rate of 1 gigabit per second.

Corresponding HBA_PORTSPEED value: HBA_PORTSPEED_1GBIT

VDS_HSF_2GBIT: The HBA port supports a transfer rate of 2 gigabits per second.

Corresponding HBA_PORTSPEED value: HBA_PORTSPEED_2GBIT

VDS_HSF_10GBIT: The HBA port supports a transfer rate of 10 gigabits per second.

Corresponding HBA_PORTSPEED value: HBA_PORTSPEED_10GBIT

VDS_HSF_4GBIT: The HBA port supports a transfer rate of 4 gigabits per second.

Corresponding HBA_PORTSPEED value: HBA_PORTSPEED_4GBIT

VDS_HSF_NOT_NEGOTIATED: The HBA port speed has not been established.

Corresponding HBA_PORTSPEED value: HBA_PORTSPEED_NOT_NEGOTIATED

2.2.2.3.1.4 VDS_PATH_STATUS

The **VDS_PATH_STATUS** enumeration defines the set of status values for a path to a storage device.

```
typedef  enum _VDS_PATH_STATUS
{
  VDS_MPS_UNKNOWN = 0x00000000,
  VDS_MPS_ONLINE = 0x00000001,
  VDS_MPS_FAILED = 0x00000005,
  VDS_MPS_STANDBY = 0x00000007
} VDS_PATH_STATUS;
```

VDS_MPS_UNKNOWN: The status of the path is unknown.

VDS_MPS_ONLINE: The path is available.

VDS_MPS_FAILED: The path is unavailable.

VDS_MPS_STANDBY: The path is on standby; it is available but will not be used unless other paths fail.

2.2.2.3.2 Structures

2.2.2.3.2.1 VDS_WWN

The **VDS_WWN** structure defines a worldwide name (WWN). This structure corresponds to the HBA_WWN structure, as specified in [HBAAPI], which also defines the WWN term.<19>

```
typedef struct _VDS_WWN {
  unsigned char rguchWwn[8];
} VDS_WWN;
```

rguchWwn: An array of 8 bytes that specifies the 64-bit WWN value. The first element of the array is the most significant byte of the WWN, and the most significant bit of that byte is the most significant bit of the WWN.

2.2.2.3.2.2 VDS_HBAPORT_PROP

The **VDS_HBAPORT_PROP** structure defines the properties of an HBA port.<20>

```
typedef struct _VDS_HBAPORT_PROP {
  VDS_OBJECT_ID id;
  VDS_WWN wwnNode;
  VDS_WWN wwnPort;
  VDS_HBAPORT_TYPE type;
  VDS_HBAPORT_STATUS status;
  unsigned long ulPortSpeed;
  unsigned long ulSupportedPortSpeed;
} VDS_HBAPORT_PROP;
```

id: The VDS object ID of the HBA port object.

wwnNode: The node WWN for the HBA port.

wwnPort: The port WWN of the HBA port.

type: The type of the HBA port that **VDS_HBAPORT_TYPE** enumerates.

status: The status of the HBA port that **VDS_HBAPORT_STATUS** enumerates.

ulPortSpeed: The speed of the HBA port that **VDS_HBAPORT_SPEED_FLAG** enumerates. Only one bit may be set in this bitmask.

ulSupportedPortSpeed: The combination of values, by using a bitwise OR operator, from the **VDS_HBAPORT_SPEED_FLAG** enumeration that describes the set of supported speeds of the HBA port.

2.2.2.4 IVdsIscsiInitiatorAdapter Data Types

This section lists data types that are used exclusively by the IVdsIscsiInitiatorAdapter methods of the **IVdsIscsiInitiatorAdapter** interface.

2.2.2.4.1 Structures

2.2.2.4.1.1 VDS_ISCSI_INITIATOR_ADAPTER_PROP

The **VDS_ISCSI_INITIATOR_ADAPTER_PROP** structure defines the properties of an iSCSI initiator adapter.<21>

```
typedef struct _VDS_ISCSI_INITIATOR_ADAPTER_PROP {
  VDS_OBJECT_ID id;
  [string] WCHAR* pwszName;
} VDS_ISCSI_INITIATOR_ADAPTER_PROP;
```

id: The VDS object ID of the initiator adapter object.

pwszName: A human-readable, null-terminated Unicode string that is the name of the initiator adapter.

2.2.2.5 IVdsIscsiInitiatorPortal Data Types

This section lists data types that are used exclusively by the IVdsIscsiInitiatorPortal methods of the **IVdsIscsiInitiatorPortal** interface.

2.2.2.5.1 Enumerations

2.2.2.5.1.1 VDS_IPADDRESS_TYPE

The **VDS_IPADDRESS_TYPE** enumeration defines the set of valid types for an IP address. These type values are used in the type member of the **VDS_IPADDRESS** structure.<22>

```
typedef  enum _VDS_IPADDRESS_TYPE
{
  VDS_IPT_TEXT = 0x00000000,
  VDS_IPT_IPV4 = 0x00000001,
  VDS_IPT_IPV6 = 0x00000002,
  VDS_IPT_EMPTY = 0x00000003
} VDS_IPADDRESS_TYPE;
```

VDS_IPT_TEXT: The IP address is a text string.

VDS_IPT_IPV4: The IP address is an IPv4 address.

VDS_IPT_IPV6: The IP address is an IPv6 address.

VDS_IPT_EMPTY: An IP address is not specified.

2.2.2.5.2 Structures

2.2.2.5.2.1 VDS_IPADDRESS

The **VDS_IPADDRESS** structure defines an IP address and port.<23>

```
typedef struct _VDS_IPADDRESS {
  VDS_IPADDRESS_TYPE type;
  unsigned long ipv4Address;
  unsigned char ipv6Address[16];
  unsigned long ulIpv6FlowInfo;
  unsigned long ulIpv6ScopeId;
  WCHAR wszTextAddress[257];
  unsigned long ulPort;
} VDS_IPADDRESS;
```

type: The type of address as enumerated by **VDS_IPADDRESS_TYPE**.

ipv4Address: If the type member is VDS_IPT_IPV4, this member contains the binary IPv4 address in network byte order. The field 3 (last octet) byte value is contained in bits 0 through 7. The byte value for field 2 is contained in bits 8 through 15. The byte value for field 1 is contained in bits 16 through 23. The byte value for field 0 is contained in bits 24 through 31. Otherwise, this value is ignored.

ipv6Address: If the type member is VDS_IPT_IPV6, this member contains the binary IPv6 address in network byte order. Otherwise, this value is ignored.

ulIpv6FlowInfo: If the type member is VDS_IPT_IPV6, this member contains the flow information as defined in IPv6. Otherwise, this value is ignored.

ulIpv6ScopeId: If the type member is VDS_IPT_IPV6, this member contains the scope ID as defined in IPv6. Otherwise, this value is ignored.

wszTextAddress: If the type member is VDS_IPT_TEXT, this member contains the null-terminated Unicode text address, which is either a DNS address, an IPv4 dotted address, or an IPv6 hexadecimal address. Otherwise, this value is ignored.

ulPort: If the type member is VDS_IPT_IPV4, VDS_IPT_IPV6, or VDS_IPT_TEXT, this member contains the TCP port number. Otherwise, this value is ignored.

2.2.2.5.2.2 VDS_ISCSI_INITIATOR_PORTAL_PROP

The **VDS_ISCSI_INITIATOR_PORTAL_PROP** structure defines the properties of an iSCSI initiator portal.<24>

```
typedef struct _VDS_ISCSI_INITIATOR_PORTAL_PROP {
```

```
  VDS_OBJECT_ID id;
  VDS_IPADDRESS address;
  unsigned long ulPortIndex;
} VDS_ISCSI_INITIATOR_PORTAL_PROP;
```

id: The VDS object ID of the initiator portal object.

address: The IP address and port of the portal.

ulPortIndex: The port index that the iSCSI initiators service assigned to the portal.

2.2.2.6 IVdsProvider Data Types

This section lists data types that are used exclusively by the **IVdsProvider** methods.

2.2.2.6.1 Enumerations

2.2.2.6.1.1 VDS_PROVIDER_TYPE

The **VDS_PROVIDER_TYPE** enumeration defines the set of valid types for a provider.

```
typedef  enum _VDS_PROVIDER_TYPE
{
  VDS_PT_UNKNOWN = 0x00000000,
  VDS_PT_SOFTWARE = 0x00000001,
  VDS_PT_HARDWARE = 0x00000002
} VDS_PROVIDER_TYPE;
```

VDS_PT_UNKNOWN: The type is neither a software nor a hardware provider.

VDS_PT_SOFTWARE: The type indicates a program that is responsible for volume management.

VDS_PT_HARDWARE: The type indicates a program that is responsible for aspects of hardware storage management.

2.2.2.6.1.2 VDS_PROVIDER_FLAG

The **VDS_PROVIDER_FLAG** enumeration defines the set of valid flags for a provider object.

```
typedef  enum _VDS_PROVIDER_FLAG
{
  VDS_PF_DYNAMIC = 0x00000001,
  VDS_PF_ONE_DISK_ONLY_PER_PACK = 0x00000004,
  VDS_PF_ONE_PACK_ONLINE_ONLY = 0x00000008,
  VDS_PF_VOLUME_SPACE_MUST_BE_CONTIGUOUS = 0x00000010,
  VDS_PF_SUPPORT_DYNAMIC = 0x80000000,
  VDS_PF_SUPPORT_FAULT_TOLERANT = 0x40000000,
  VDS_PF_SUPPORT_DYNAMIC_1394 = 0x20000000
} VDS_PROVIDER_FLAG;
```

VDS_PF_DYNAMIC: If set, all disks that the current provider manages are dynamic. This flag MUST be set only by a **dynamic provider**. By definition, dynamic providers manage only dynamic disks.

VDS_PF_ONE_DISK_ONLY_PER_PACK: If set, the provider supports single **disk packs** only. Typically, the **basic provider** sets this flag to simulate a disk pack that has one disk.

VDS_PF_ONE_PACK_ONLINE_ONLY: If set, the dynamic provider supports online status for only one pack at a time.

VDS_PF_VOLUME_SPACE_MUST_BE_CONTIGUOUS: If set, all volumes that this provider manages must have contiguous space. This flag applies to the basic provider only.

VDS_PF_SUPPORT_DYNAMIC: If set, the provider supports managing dynamic disks. This flag MUST be set only by the dynamic provider on systems that support dynamic disks.

VDS_PF_SUPPORT_FAULT_TOLERANT: If set, the provider supports fault-tolerant disks. This flag MUST be set only by the dynamic provider on systems that support fault-tolerant volumes.

VDS_PF_SUPPORT_DYNAMIC_1394: If set, the provider supports IEEE 1394 dynamic disks. This flag MUST be set only by the dynamic provider on systems that support IEEE 1394 dynamic disks.

2.2.2.6.2 Structures

2.2.2.6.2.1 VDS_PROVIDER_PROP

The **VDS_PROVIDER_PROP** structure provides information about provider properties.

```
typedef struct _VDS_PROVIDER_PROP {
  VDS_OBJECT_ID id;
  [string] WCHAR* pwszName;
  GUID guidVersionId;
  [string] WCHAR* pwszVersion;
  VDS_PROVIDER_TYPE type;
  unsigned long ulFlags;
  unsigned long ulStripeSizeFlags;
  short sRebuildPriority;
} VDS_PROVIDER_PROP;
```

id: The VDS object ID of the provider object.

pwszName: The null-terminated Unicode name of the provider.

guidVersionId: The version **GUID** of the provider. This GUID MUST be unique to each version of the provider.

pwszVersion: The null-terminated Unicode version string of the provider. The convention for this string is <major version number>.<minor version number>.

type: A value from the **VDS_PROVIDER_TYPE** enumeration that indicates the provider type.

ulFlags: A combination of any values, by using a bitwise OR operator, from the **VDS_PROVIDER_FLAG** enumeration.

ulStripeSizeFlags: Stripe size that the provider supports, which MUST be a power of 2. Each bit in the 32-bit integer represents a size that the provider supports. For example, if the nth bit is set, the provider supports a stripe size of 2^n. This parameter is used only for software providers. The basic provider sets this value to zero and the dynamic provider sets this value to 64K.

sRebuildPriority: The rebuild priority of all volumes that the provider manages. It specifies the regeneration order when a mirrored or RAID-5 volume requires rebuilding. Priority levels MUST be from 0 through 15. A higher value indicates a higher priority. This parameter is used only for software providers and does not apply to the basic provider.

2.2.2.7 IVdsPack Data Types

This section lists the data types that are used exclusively by the **IVdsPack** methods.

2.2.2.7.1 Enumerations

2.2.2.7.1.1 VDS_PACK_STATUS

The **VDS_PACK_STATUS** enumeration defines the set of object status values for a disk pack.

```
typedef  enum _VDS_PACK_STATUS
{
  VDS_PS_UNKNOWN = 0x00000000,
  VDS_PS_ONLINE = 0x00000001,
  VDS_PS_OFFLINE = 0x00000004
} VDS_PACK_STATUS;
```

VDS_PS_UNKNOWN: The status of the disk pack cannot be determined.

VDS_PS_ONLINE: The disk pack is available.

VDS_PS_OFFLINE: The disk pack is unavailable; the disks are not accessible.

2.2.2.7.1.2 VDS_PACK_FLAG

The **VDS_PACK_FLAG** enumeration defines the set of valid flags for a disk pack object.

```
typedef  enum _VDS_PACK_FLAG
{
  VDS_PKF_FOREIGN = 0x00000001,
  VDS_PKF_NOQUORUM = 0x00000002,
  VDS_PKF_POLICY = 0x00000004,
  VDS_PKF_CORRUPTED = 0x00000008,
  VDS_PKF_ONLINE_ERROR = 0x00000010
} VDS_PACK_FLAG;
```

VDS_PKF_FOREIGN: If set, an external disk pack is eligible for online status.

VDS_PKF_NOQUORUM: If set, a dynamic disk pack lacks the required **disk quorum**.

VDS_PKF_POLICY: If set, management policy forbids the disk pack from gaining online status.

VDS_PKF_CORRUPTED: If set, a disk pack contains a disk that has a corrupted **LDM** database.

VDS_PKF_ONLINE_ERROR: If set, a pack with sufficient disk quorum failed to achieve online status due to an error.

2.2.2.7.2 Structures

2.2.2.7.2.1 VDS_PACK_PROP

The **VDS_PACK_PROP** structure provides information about the properties of a disk pack.

```
typedef struct _VDS_PACK_PROP {
  VDS_OBJECT_ID id;
  [string] WCHAR* pwszName;
  VDS_PACK_STATUS status;
  unsigned long ulFlags;
} VDS_PACK_PROP,
 *PVDS_PACK_PROP;
```

id: The VDS object ID of the disk pack object.

pwszName: The null-terminated Unicode name of the disk pack.

status: The value from the **VDS_PACK_STATUS** enumeration that indicates the status of the disk pack.

ulFlags: A combination of any values, by using a bitwise OR operator, of the disk pack flags that are defined in the **VDS_PACK_FLAG** enumeration.

2.2.2.8 IVdsDisk Data Types

This section lists data types that are used exclusively by the **IVdsDisk** methods.

2.2.2.8.1 Enumerations

2.2.2.8.1.1 VDS_DISK_STATUS

The **VDS_DISK_STATUS** enumeration defines the status of a disk.

```
typedef  enum _VDS_DISK_STATUS
{
  VDS_DS_UNKNOWN = 0x00000000,
  VDS_DS_ONLINE = 0x00000001,
  VDS_DS_NOT_READY = 0x00000002,
  VDS_DS_NO_MEDIA = 0x00000003,
  VDS_DS_FAILED = 0x00000005,
  VDS_DS_MISSING = 0x00000006
} VDS_DISK_STATUS;
```

VDS_DS_UNKNOWN: The disk status is unknown.

VDS_DS_ONLINE: The disk is online.

VDS_DS_NOT_READY: The disk is not ready.

VDS_DS_NO_MEDIA: The disk has no media.

VDS_DS_FAILED: The disk failed.

VDS_DS_MISSING: The disk is missing; it is no longer available to the operating system.

2.2.2.8.1.2 VDS_DISK_FLAG

The **VDS_DISK_FLAG** enumeration defines the properties of a disk.

```
typedef  enum _VDS_DISK_FLAG
{
  VDS_DF_AUDIO_CD = 0x00000001,
  VDS_DF_HOTSPARE = 0x00000002,
  VDS_DF_RESERVE_CAPABLE = 0x00000004,
  VDS_DF_MASKED = 0x00000008,
  VDS_DF_STYLE_CONVERTIBLE = 0x00000010,
  VDS_DF_CLUSTERED = 0x00000020,
  VDS_DF_READ_ONLY = 0x00000040,
  VDS_DF_SYSTEM_DISK = 0x0000080,
  VDS_DF_BOOT_DISK = 0x00000100,
  VDS_DF_PAGEFILE_DISK = 0x00000200,
  VDS_DF_HIBERNATIONFILE_DISK = 0x00000400,
  VDS_DF_CRASHDUMP_DISK = 0x00000800
} VDS_DISK_FLAG;
```

VDS_DF_AUDIO_CD: The disk is an audio CD, as specified in [IEC60908].

VDS_DF_HOTSPARE: The disk is a hot spare.

VDS_DF_RESERVE_CAPABLE: The disk can be reserved for a host.

VDS_DF_MASKED: The disk is currently hidden from the host.

VDS_DF_STYLE_CONVERTIBLE: The disk is convertible between the MBR partition format and the GPT partition format.

VDS_DF_CLUSTERED: The disk is clustered.

VDS_DF_READ_ONLY: The disk is marked as read-only.

VDS_DF_SYSTEM_DISK: The disk contains the system volume.

VDS_DF_BOOT_DISK: The disk contains the boot volume.

VDS_DF_PAGEFILE_DISK: The disk contains the paging file on one of its volumes.

VDS_DF_HIBERNATIONFILE_DISK: The disk contains the hibernation file on one of its volumes.

VDS_DF_CRASHDUMP_DISK: The disk is configured to contain a crash-dump file on one of its volumes.

2.2.2.8.1.3 VDS_LUN_RESERVE_MODE

The **VDS_LUN_RESERVE_MODE** enumeration defines the sharing mode of a disk.

```
typedef  enum _VDS_LUN_RESERVE_MODE
{
  VDS_LRM_NONE = 0x00000000,
  VDS_LRM_EXCLUSIVE_RW = 0x00000001,
  VDS_LRM_EXCLUSIVE_RO = 0x00000002,
  VDS_LRM_SHARED_RO = 0x00000003,
  VDS_LRM_SHARED_RW = 0x00000004
} VDS_LUN_RESERVE_MODE;
```

VDS_LRM_NONE: The disk has no assigned sharing mode.

VDS_LRM_EXCLUSIVE_RW: The disk is reserved for exclusive access.

VDS_LRM_EXCLUSIVE_RO: The disk is available for read access.

VDS_LRM_SHARED_RO: The disk is available for shared read access.

VDS_LRM_SHARED_RW: The disk is available for shared read/write access.

2.2.2.8.2 Structures

2.2.2.8.2.1 VDS_DISK_PROP

The **VDS_DISK_PROP** structure provides the properties of a disk.

```
typedef struct _VDS_DISK_PROP {
  VDS_OBJECT_ID id;
  VDS_DISK_STATUS status;
  VDS_LUN_RESERVE_MODE ReserveMode;
  VDS_HEALTH health;
  DWORD dwDeviceType;
  DWORD dwMediaType;
  ULONGLONG ullSize;
  unsigned long ulBytesPerSector;
  unsigned long ulSectorsPerTrack;
  unsigned long ulTracksPerCylinder;
  unsigned long ulFlags;
  VDS_STORAGE_BUS_TYPE BusType;
  VDS_PARTITION_STYLE PartitionStyle;
  [switch_is(PartitionStyle)] union {
    [case(VDS_PST_MBR)]
      DWORD dwSignature;
    [case(VDS_PST_GPT)]
      GUID DiskGuid;
    [default]
      ;
  };
  [string] WCHAR* pwszDiskAddress;
  [string] WCHAR* pwszName;
  [string] WCHAR* pwszFriendlyName;
  [string] WCHAR* pwszAdaptorName;
  [string] WCHAR* pwszDevicePath;
} VDS_DISK_PROP,
 *PVDS_DISK_PROP;
```

id: The VDS object ID of the disk object.

status: The value from the **VDS_DISK_STATUS** enumeration that indicates the disk status.

ReserveMode: The value from the **VDS_LUN_RESERVE_MODE** enumeration that indicates the sharing mode of the disk.

health: The value from the **VDS_HEALTH** enumeration that indicates the health of the disk.

dwDeviceType: The device type of the disk. Note that this value refers to the **disk type** and not the drive type. Thus, if there is CD media in a DVD/CD drive, it is identified as FILE_DEVICE_CD_ROM; however, DVD media in the same drive is identified as FILE_DEVICE_DVD. This field can have the following values:

Value	Meaning
FILE_DEVICE_CD_ROM 0x00000002	The device is a CD-ROM.
FILE_DEVICE_DISK 0x00000007	The device is a hard disk.
FILE_DEVICE_DVD 0x00000033	The device is a DVD.

dwMediaType: The media type of the disk; it can have the following values:

Value	Meaning
Unknown 0x00000000	The disk media type is unknown.
RemovableMedia 0x0000000B	The disk media is removable.
FixedMedia 0x0000000C	The disk media is fixed.

ullSize: The size of the disk, in bytes.

ulBytesPerSector: The size of the sectors for the disk, in bytes.

ulSectorsPerTrack: The number of sectors per **track** on the disk.

ulTracksPerCylinder: The number of tracks per **cylinder** on the disk.

ulFlags: The combination of any values, by using a bitwise OR operator, that are defined in the **VDS_DISK_FLAG** enumeration.

BusType: The value from the **VDS_STORAGE_BUS_TYPE** enumeration that indicates the type of bus where the disk resides.

PartitionStyle: The value from the **VDS_PARTITION_STYLE** enumeration that indicates the partitioning format of the disk.

dwSignature: The MBR disk signature of the disk.

DiskGuid: The GUID in the GPT that identifies the disk.

pwszDiskAddress: The null-terminated Unicode address of the disk, if the disk uses a SCSI-like address; otherwise, NULL. If present, a client can use this property to determine the port number, bus, target number and LUN of the disk.

pwszName: The null-terminated Unicode name that the operating system uses to identify the disk. If present, a client can use this property to determine the disk's PNP device number. For a hard disk, this name has the format \\?\PhysicalDriveN, where N signifies the device number of the disk. For a DVD/CD drive, this name has the format \\?\CdRomN, where N signifies the device number of the DVD/CD drive. A client can use this property to identify the disk.

pwszFriendlyName: The null-terminated Unicode friendly (human-readable) name of the disk as assigned by the operating system. This property MAY be NULL. If present, a client can use this property to display a human-readable name of the disk.

pwszAdaptorName: The null-terminated Unicode name that the operating system assigns to the adapter to which the disk is attached. This property MAY be NULL. If present, a client can use this property to display the adaptor name of the disk.

pwszDevicePath: The null-terminated Unicode device path that the operating system uses to identify the device for the disk. This property MAY be NULL. If present, a client can use this property to display the device path of the disk.<25>

2.2.2.9 IVdsAdvancedDisk Data Types

This section lists data types that are used exclusively by the **IVdsAdvancedDisk** methods.

2.2.2.9.1 Structures

2.2.2.9.1.1 CHANGE_ATTRIBUTES_PARAMETERS

The **CHANGE_ATTRIBUTES_PARAMETERS** structure describes the attributes to change on a partition.

```
typedef struct _CHANGE_ATTRIBUTES_PARAMETERS {
  VDS_PARTITION_STYLE style;
  [switch_is(style)] union {
    [case(VDS_PST_MBR)]
      struct {
      boolean bootIndicator;
    } MbrPartInfo;
    [case(VDS_PST_GPT)]
      struct {
      ULONGLONG attributes;
    } GptPartInfo;
    [default]
      ;
  };
} CHANGE_ATTRIBUTES_PARAMETERS;
```

style: The value from the **VDS_PARTITION_STYLE** enumeration that describes the partition format of the disk. If the disk partitioning format is MBR, the only value that MAY be changed is the bootIndicator. If the disk partitioning format is GPT, the only value that MAY be changed is the GPT attribute.

MbrPartInfo: Contains information for an MBR partition.

bootIndicator: The Boolean value that indicates whether the partition is bootable.

GptPartInfo: Contains information for a partition in a GPT.

attributes: The bitwise OR operator of attributes to change; it can have a combination of the following values:

Value	Meaning
GPT_ATTRIBUTE_PLATFORM_REQUIRED 0x0000000000000001	Partition is required for the platform to function properly.<26>
GPT_BASIC_DATA_ATTRIBUTE_READ_ONLY 0x1000000000000000	The partition can be read from but not written to. Used only with the basic data partition type.
GPT_BASIC_DATA_ATTRIBUTE_HIDDEN 0x4000000000000000	The partition is hidden and is not mounted. Used only with the basic data partition type.
GPT_BASIC_DATA_ATTRIBUTE_NO_DRIVE_LETTER 0x8000000000000000	The partition does not receive a drive letter by default when moving the disk to another machine. Used only with the basic data partition type.

2.2.2.10 IVdsAdvancedDisk2 Data Types

This section lists data types that are used exclusively by the **IVdsAdvancedDisk2** methods.

2.2.2.10.1 Structures

2.2.2.10.1.1 CHANGE_PARTITION_TYPE_PARAMETERS

The **CHANGE_PARTITION_TYPE_PARAMETERS** structure describes parameters to use when changing a partition type.<27>

```
typedef struct _CHANGE_PARTITION_TYPE_PARAMETERS {
  VDS_PARTITION_STYLE style;
  [switch_is(style)] union {
    [case(VDS_PST_MBR)]
      struct {
      byte partitionType;
    } MbrPartInfo;
    [case(VDS_PST_GPT)]
      struct {
      GUID partitionType;
    } GptPartInfo;
    [default]
      ;
  };
} CHANGE_PARTITION_TYPE_PARAMETERS;
```

style: A value from the VDS_PARITION_STYLE enumeration that describes the disk partition format.

MbrPartInfo: Contains information for an MBR partition.

partitionType: The byte value indicating the partition type to change the partition to.

GptPartInfo: Contains information for the partition of a GPT.

partitionType: The GUID indicating the partition type to change the partition to.<28>

2.2.2.11 IVdsVolume Data Types

This section lists data types that are used exclusively by the **IVdsVolume** methods.

2.2.2.11.1 Enumerations

2.2.2.11.1.1 VDS_VOLUME_STATUS

The **VDS_VOLUME_STATUS** enumeration defines the set of object status values for a volume.

```
typedef  enum _VDS_VOLUME_STATUS
{
  VDS_VS_UNKNOWN = 0x00000000,
  VDS_VS_ONLINE = 0x00000001,
  VDS_VS_NO_MEDIA = 0x00000003,
  VDS_VS_FAILED = 0x00000005
} VDS_VOLUME_STATUS;
```

VDS_VS_UNKNOWN: The status of the volume is unknown.

VDS_VS_ONLINE: The volume is available.

VDS_VS_NO_MEDIA: The volume belongs to a removable media device, such as a CD-ROM or DVD-ROM drive, but the device does not currently have media in the drive.

VDS_VS_FAILED: The volume is unavailable.

2.2.2.11.1.2 VDS_VOLUME_FLAG

The **VDS_VOLUME_FLAG** enumeration defines the set of valid flags for a volume object.

```
typedef  enum _VDS_VOLUME_FLAG
{
  VDS_VF_SYSTEM_VOLUME = 0x00000001,
  VDS_VF_BOOT_VOLUME = 0x00000002,
  VDS_VF_ACTIVE = 0x00000004,
  VDS_VF_READONLY = 0x00000008,
  VDS_VF_HIDDEN = 0x00000010,
  VDS_VF_CAN_EXTEND = 0x00000020,
  VDS_VF_CAN_SHRINK = 0x00000040,
  VDS_VF_PAGEFILE = 0x00000080,
```

```
  VDS_VF_HIBERNATION = 0x00000100,
  VDS_VF_CRASHDUMP = 0x00000200,
  VDS_VF_INSTALLABLE = 0x00000400,
  VDS_VF_LBN_REMAP_ENABLED = 0x00000800,
  VDS_VF_FORMATTING = 0x00001000,
  VDS_VF_NOT_FORMATTABLE = 0x00002000,
  VDS_VF_NTFS_NOT_SUPPORTED = 0x00004000,
  VDS_VF_FAT32_NOT_SUPPORTED = 0x00008000,
  VDS_VF_FAT_NOT_SUPPORTED = 0x00010000,
  VDS_VF_NO_DEFAULT_DRIVE_LETTER = 0x00020000,
  VDS_VF_PERMANENTLY_DISMOUNTED = 0x00040000,
  VDS_VF_PERMANENT_DISMOUNT_SUPPORTED = 0x00080000,
  VDS_VF_SHADOW_COPY = 0x00100000,
  VDS_VF_FVE_ENABLED = 0x00200000
} VDS_VOLUME_FLAG;
```

VDS_VF_SYSTEM_VOLUME: If set, the volume is a **system volume**. It contains the **boot loader** that is used to invoke the operating system on the **boot volume**.

VDS_VF_BOOT_VOLUME: If set, the volume is a boot volume that contains the operating system.

VDS_VF_ACTIVE: If set, the volume is an **active volume**. It can become the system volume at system startup if the BIOS is configured to select that disk for startup.

VDS_VF_READONLY: If set, the volume can be read from but not written to.

VDS_VF_HIDDEN: If set, the volume does not automatically get assigned mount points or drive letters that can be used to access the volume.

VDS_VF_CAN_EXTEND: If set, the volume size can be extended.

VDS_VF_CAN_SHRINK: If set, the volume size can be reduced.

VDS_VF_PAGEFILE: If this flag is set, the volume contains a **page file**.

VDS_VF_HIBERNATION: If set, the volume holds the files that are used when the system hibernates.

VDS_VF_CRASHDUMP: If set, the volume acts as a crash-dump device.

VDS_VF_INSTALLABLE: If set, callers can use the volume to install an operating system.

VDS_VF_LBN_REMAP_ENABLED: If set, VDS can dynamically change the position of the volume on the disk.<29>

VDS_VF_FORMATTING: If set, the volume is being formatted.

VDS_VF_NOT_FORMATTABLE: If set, the volume cannot be formatted. This flag applies to small portable memory devices, removable devices, CD-ROM devices, and DVD devices.

VDS_VF_NTFS_NOT_SUPPORTED: If set, the volume does not support the NTFS file system, but can support other file systems. This flag may apply to small portable memory devices, removable devices, CD-ROM devices, and DVD devices.

VDS_VF_FAT32_NOT_SUPPORTED: If set, the volume does not support FAT32. This flag MAY apply to small portable memory devices, removable devices, CD-ROM devices, and DVD devices.

VDS_VF_FAT_NOT_SUPPORTED: If set, the volume does not support FAT. This flag MAY apply to small portable memory devices, removable devices, CD-ROM devices, and DVD devices.

VDS_VF_NO_DEFAULT_DRIVE_LETTER: If set, the operating system does not automatically assign a drive letter when the volume first appears. When cleared, the operating system assigns a drive letter to the partition under some conditions. Callers can set and clear this flag on all but **dynamic volumes**. On dynamic volumes, this flag is always set. For basic GPT volumes, assigning or removing a drive letter toggles this flag.

VDS_VF_PERMANENTLY_DISMOUNTED: If set, the volume is unavailable and requires a mount-point assignment. VDS sets this flag after the caller invokes the **IVdsVolumeMF::Dismount** method, setting the *bForce* and *bPermanent* parameters to TRUE.

VDS_VF_PERMANENT_DISMOUNT_SUPPORTED: If set, the volume supports *bPermanent* for the **IVdsVolumeMF::Dismount** method.

VDS_VF_SHADOW_COPY: If set, the volume is a **shadow copy** of another volume. This flag is set when the **snapshot** is taken, and it is cleared when the snapshot is broken from the original volume. The **VDS_VF_SHADOW_COPY** flag is an indication for software-like file system filter drivers (for example, antivirus) to avoid attaching to the volume. Applications can use the attribute to differentiate snapshots from production volumes. Applications that create a Fast Recovery where a shadow copy LUN is made into a non-snapshot by clearing the read-only and hidden bit, will need to clear this bit as well.

VDS_VF_FVE_ENABLED: The volume is encrypted with **BitLocker** full-volume encryption.

2.2.2.11.2 Structures

2.2.2.11.2.1 VDS_VOLUME_PROP

The **VDS_VOLUME_PROP** structure that provides the properties of a volume.

```
typedef struct _VDS_VOLUME_PROP {
  VDS_OBJECT_ID id;
  VDS_VOLUME_TYPE type;
  VDS_VOLUME_STATUS status;
  VDS_HEALTH health;
  VDS_TRANSITION_STATE TransitionState;
  ULONGLONG ullSize;
  unsigned long ulFlags;
  VDS_FILE_SYSTEM_TYPE RecommendedFileSystemType;
  [string] WCHAR* pwszName;
} VDS_VOLUME_PROP,
 *PVDS_VOLUME_PROP;
```

id: The VDS object ID of the volume object.

type: The value from the **VDS_VOLUME_TYPE** enumeration that defines the type of the volume.

status: The value from the **VDS_VOLUME_STATUS** enumeration that defines the status of the volume.

health: The value from the **VDS_HEALTH** enumeration that defines the health of the volume.

TransitionState: The value from the **VDS_TRANSITION_STATE** enumeration that defines the configuration stability of the volume.

ullSize: The size of the volume, in bytes.

ulFlags: The combination of any values by using the bitwise OR operator of volume flags from the **VDS_VOLUME_FLAG** enumeration.

RecommendedFileSystemType: The value from the **VDS_FILE_SYSTEM_TYPE** enumeration that defines the recommended file system type for the volume.

pwszName: The null-terminated Unicode name that the operating system uses to identify the volume.

2.2.2.12 IVdsVolumeMF Data Types

This section lists data types that are used exclusively by the **IVdsVolumeMF** methods.

2.2.2.12.1 Data Types

2.2.2.12.1.1 MAX_PATH

Constant/value	Description
MAX_PATH 0x00000104	The maximum character length of a path.

2.2.2.12.2 Structures

2.2.2.12.2.1 VDS_REPARSE_POINT_PROP

The **VDS_REPARSE_POINT_PROP** structure defines the reparse point properties of the mount point to a volume object.

```
typedef struct VDS_REPARSE_POINT_PROP {
  VDS_OBJECT_ID SourceVolumeId;
  [string] WCHAR* pwszPath;
} VDS_REPARSE_POINT_PROP,
 *PVDS_REPARSE_POINT_PROP;
```

SourceVolumeId: The VDS object ID of the volume object that the reparse point refers to.

pwszPath: The null-terminated Unicode path of the reparse point. The path does not contain a drive letter. For example, "\mount".

2.2.2.13 IVdsVolumePlex Data Types

This section lists data types that are used exclusively by the **IVdsVolumePlex** methods.

2.2.2.13.1 Enumeration

2.2.2.13.1.1 VDS_VOLUME_PLEX_TYPE

The **VDS_VOLUME_PLEX_TYPE** enumeration defines the set of valid types for a volume plex.

```
typedef  enum _VDS_VOLUME_PLEX_TYPE
{
  VDS_VPT_UNKNOWN = 0x00000000,
  VDS_VPT_SIMPLE = 0x0000000A,
  VDS_VPT_SPAN = 0x0000000B,
  VDS_VPT_STRIPE = 0x0000000C,
  VDS_VPT_PARITY = 0x0000000E
} VDS_VOLUME_PLEX_TYPE;
```

VDS_VPT_UNKNOWN: The volume plex type is unknown.

VDS_VPT_SIMPLE: The plex type is simple; it is composed of extents from exactly one disk.

VDS_VPT_SPAN: The plex type is spanned; it is composed of extents from more than one disk.

VDS_VPT_STRIPE: The plex type is striped, which is equivalent to RAID-0.

VDS_VPT_PARITY: The plex type is striped with parity, which accounts for RAID levels 3, 4, 5, and 6.

2.2.2.13.1.2 VDS_VOLUME_PLEX_STATUS

The **VDS_VOLUME_PLEX_STATUS** enumeration defines the set of object status values for a volume plex.

```
typedef  enum _VDS_VOLUME_PLEX_STATUS
{
  VDS_VPS_UNKNOWN = 0x00000000,
  VDS_VPS_ONLINE = 0x00000001,
  VDS_VPS_NO_MEDIA = 0x00000003,
  VDS_VPS_FAILED = 0x00000005
} VDS_VOLUME_PLEX_STATUS;
```

VDS_VPS_UNKNOWN: The status of the volume plex is unknown.

VDS_VPS_ONLINE: The volume plex is available.

VDS_VPS_NO_MEDIA: The volume plex has no media.

VDS_VPS_FAILED: The volume plex is unavailable.

2.2.2.13.2 Structures

2.2.2.13.2.1 VDS_VOLUME_PLEX_PROP

The **VDS_VOLUME_PLEX_PROP** structure provides information about the properties of a volume plex.

```
typedef struct _VDS_VOLUME_PLEX_PROP {
  VDS_OBJECT_ID id;
  VDS_VOLUME_PLEX_TYPE type;
  VDS_VOLUME_PLEX_STATUS status;
  VDS_HEALTH health;
  VDS_TRANSITION_STATE TransitionState;
  ULONGLONG ullSize;
  unsigned long ulStripeSize;
  unsigned long ulNumberOfMembers;
} VDS_VOLUME_PLEX_PROP,
 *PVDS_VOLUME_PLEX_PROP;
```

id: The GUID of the plex object.

type: The plex type that is enumerated by **VDS_VOLUME_PLEX_TYPE**. The type of the plex need not match that of the volume to which it belongs. For example, a mirrored RAID-1 volume can be composed of plexes that are simple (composed of extents from exactly one disk).

status: The status of the plex object that is enumerated by **VDS_VOLUME_PLEX_STATUS**. The status of the plex need not match that of the volume to which it belongs. For example, a volume plex may have a failed status (VDS_VPS_FAILED), but if the volume is fault-tolerant and its other plexes are online (VDS_VPS_ONLINE), the volume will still be online (VDS_VS_ONLINE).

health: Value from the **VDS_HEALTH** enumeration that defines the health of the volume. The health of the plex need not match that of the volume to which it belongs. For instance, a volume's plex may have failed health (VDS_H_FAILED), but if the volume is a mirror volume (RAID-1) and its other plexes are healthy (VDS_H_HEALTHY), the volume will have failed redundancy health (VDS_H_FAILED_REDUNDANCY).

TransitionState: Value from the **VDS_TRANSITION_STATE** enumeration that defines the configuration stability of the plex. The TransitionState of the plex matches the TransitionState of the volume to which it belongs.

ullSize: The size of the plex, in bytes. The size can be equal to, or greater than, that of the volume to which it belongs. The plex cannot be smaller than the volume.

ulStripeSize: The stripe interleave size, in bytes. This member applies only for plexes of type VDS_VPT_STRIPE (striped) and VDS_VPT_PARITY (striped with parity).

ulNumberOfMembers: The number of members (**RAID columns**) in the volume plex.

3 Protocol Details

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

3.1 Interfaces

All VDS interfaces that are listed in this section inherit the IUnknown interface. For all VDS interfaces, method **opnum** field values start with 3; opnum values 0, 1, and 2 represent the IUnknown::QueryInterface, IUnknown::AddRef, and IUnknown::Release methods, respectively. For more information, see [MS-DCOM].

The interfaces in this section are listed in the following order:

- Generic object interfaces: sections Enumeration Object Interfaces through Asynchronous Operation Object Interfaces
- Service object interfaces: sections Service Object Interfaces through HBA Port Object Interfaces.
- Provider interfaces: Provider Object Interfaces.
- Pack interfaces: section Pack Object Interfaces.
- Disk interfaces: section Disk Object Interfaces.
- Volume interfaces: section Volume Object Interfaces.
- Volume plex interfaces: section Volume Plex Object Interfaces.

This order reflects the logical hierarchy of objects in VDS. For more information, see section 3.3.1.

To retrieve the interfaces of a particular object, call the QueryInterface method on the DCOM IUnknown interfaces of the object. For more information, see [MS-DCOM].

Unless otherwise specified, all methods MUST return zero if they are successful, or an implementation-specific nonzero error code if they fail. Unless otherwise specified, client implementations of this protocol MUST NOT take any action on an error code, but rather, they return the error to the invoking application.

3.1.1 Enumeration Object Interfaces

This section includes interfaces that are used to interact with enumeration objects (enum objects) on the server.

Enumeration objects are returned from methods of other interfaces and are used to enumerate through a set of VDS objects of a specified type. The type of object that is enumerated depends on the interface and method from which the enumeration object was returned.

Objects can be HBA ports, initiator adapters, initiator portals, providers, packs, disks, volumes, or volume plexes.

3.1.1.1 IEnumVdsObject Interface

The **IEnumVdsObject** interface enumerates through a set of VDS objects.

The UUID for this interface is {118610B7-8D94-4030-B5B8-500889788E4E}.

The **IEnumVdsObject** methods are specified in section 3.3.5.2.1.

Methods in RPC Opnum Order

Method	Description
Next	Returns a specified number of objects in the enumeration. It begins from the current point. Opnum: 3
Skip	Skips a specified number of objects in the enumeration. Opnum: 4
Reset	Resets the enumerator to the beginning of the collection. Opnum: 5
Clone	Creates a new enumeration that has the same state as the current enumeration. Opnum: 6

All methods MUST NOT throw exceptions.

3.1.2 Callback Object Interfaces

This section includes interfaces that the server uses to interact with the callback object on the client.

3.1.2.1 IVdsAdviseSink Interface

The client implements the **IVdsAdviseSink** interface in order to receive notification of VDS object changes.

The UUID for this interface is {8326CD1D-CF59-4936-B786-5EFC08798E25}.

IVdsAdviseSink methods are specified in section 3.2.4.3.

Methods in RPC Opnum Order

Method	Description
OnNotify	Passes notifications from VDS to applications. Opnum: 3

All methods MUST NOT throw exceptions.

3.1.3 Asynchronous Operation Object Interfaces

This section includes interfaces that are used to interact with **asynchronous operation objects (async objects)** on the server.

3.1.3.1 IVdsAsync Interface

The **IVdsAsync** interface manages asynchronous operations. Methods that initiate asynchronous operations return a pointer to an **IVdsAsync** interface, allowing the caller to optionally cancel, wait for, or query the status of the asynchronous operation.

The UUID for this interface is {D5D23B6D-5A55-4492-9889-397A3C2D2DBC}.

The **IVdsAsync** methods are specified in section 3.3.5.2.2.

Methods in RPC Opnum Order

Method	Description
Cancel	Cancels the asynchronous operation. This method has no parameters. Opnum: 3
Wait	Blocks and returns when the asynchronous operation has either finished successfully or failed. Opnum: 4
QueryStatus	Retrieves the status of the asynchronous operation. Opnum: 5

All methods MUST NOT throw exceptions.

3.1.4 Service Loader Interfaces

This section includes the interfaces that are used to load VDS service objects on the server.

3.1.4.1 IVdsServiceLoader Interface

Servers implement the **IVdsServiceLoader** interface, which can be used by clients to load the VDS service object on both local and remote machines.

The UUID for this interface is {E0393303-90D4-4A97-AB71-E9B671EE2729}.

The **IVdsServiceLoader** methods are specified in section 3.3.5.2.3.

Methods in RPC Opnum Order

Method	Description
LoadService	Loads the VDS service on the machine that is specified by an input parameter and returns a pointer to the **IVdsService interface**. Opnum: 3

3.1.5 Service Object Interfaces

This section includes interfaces that are used to interact with the VDS service object on the server.

3.1.5.1 IVdsService Interface

Servers implement the **IVdsService** interface in order to support storage management.

The UUID for this interface is {0818A8EF-9BA9-40D8-A6F9-E22833CC771E}.

The IVdsService methods are specified in section 3.3.5.2.4. A method is not listed for opnum 7 because the VDS Protocol does not use it.<30>

Methods in RPC Opnum Order

Method	Description
IsServiceReady	Determines whether a service is finished initializing. Opnum: 3
WaitForServiceReady	Waits for VDS initialization to complete, and then returns the status of the VDS initialization in the **HRESULT**. Opnum: 4
GetProperties	Retrieves the properties of the service that is represented by the object that exposes this interface and method. Opnum: 5
QueryProviders	Enumerates the providers of the server. Opnum: 6
Opnum07NotUsedOnWire	Reserved for local use. Opnum: 7
QueryUnallocatedDisks	Enumerates the unallocated disks on the server. Opnum: 8
GetObject	Retrieves an IUnknown pointer to a specified object. Opnum: 9
QueryDriveLetters	Enumerates the drive letters of the server. Opnum: 10
QueryFileSystemTypes	Returns property details for all file systems that are known to VDS. Opnum: 11
Reenumerate	Discovers newly added and newly removed disks and returns the status of the operation in the HRESULT. Opnum: 12
Refresh	Refreshes the ownership and layout of disks on the server. Opnum: 13
CleanupObsoleteMountPoints	Removes any mount points that point to volumes that no longer exist. Opnum: 14
Advise	Registers a notification callback with the server. Clients pass the callback object to the server to receive notifications. Opnum: 15
Unadvise	Unregisters a client from notification of changes to storage objects by the server. Opnum: 16
Reboot	Restarts the computer on which the server is running. Opnum: 17
SetFlags	Assigns property flags to the server. Opnum: 18

Method	Description
ClearFlags	Clears property flags from the service. Opnum: 19

All methods MUST NOT throw exceptions.

In the previous table, the term "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined<31>because it does not affect interoperability.

3.1.5.2 IVdsServiceInitialization Interface

The **IVdsServiceInitialization** interface is implemented by VDS and is used by clients to start initialization of the service.

The UUID for this interface is {4afc3636-db01-4052-80c3-03bbcb8d3c69}.

The **IVdsServiceInitialization** methods are specified in section 3.3.5.2.5.

Methods in RPC Opnum Order

Method	Description
Initialize	Starts the initialization of the server. Opnum: 3

All methods MUST NOT throw exceptions.

3.1.5.3 IVdsServiceUninstallDisk Interface

The service object implements the **IVdsServiceUninstallDisk** interface in order to provide a way to query VDS for disks that correspond to particular LUN information structures and to remove these disks and the volumes wholly or partially contained in them.

The UUID for this interface is {B6B22DA8-F903-4BE7-B492-C09D875AC9DA}.

The **IVdsServiceUninstallDisk** methods are specified in section 3.3.5.2.6.<32>

Methods in RPC Opnum Order

Method	Description
GetDiskIdFromLunInfo	Retrieves the VDS object ID of a disk that corresponds to a specified LUN information structure. Opnum: 3
UninstallDisks	Uninstalls a specific set of disks when it is given a list of the VDS object IDs for the disks. Opnum: 4

All methods MUST NOT throw exceptions.

3.1.5.4 IVdsServiceHba Interface

The **IVdsServiceHba** interface provides a method to query HBA ports on the server.<33>

The UUID for this interface is {0AC13689-3134-47C6-A17C-4669216801BE}.

The **IVdsServiceHba** methods are specified in section 3.3.5.2.7.

Methods in RPC Opnum Order

Method	Description
QueryHbaPorts	Returns an **IEnumVdsObject** enumeration object that contains a list of the HBA ports that are known to VDS on the local system. Opnum: 3

All methods MUST NOT throw exceptions.

3.1.5.5 IVdsServiceIscsi Interface

The **IVdsServiceIscsi** interface provides methods to interact with the iSCSI initiators service on the server. It includes the ability to set CHAP security settings and to log into targets.<34>

The UUID for this interface is {14FBE036-3ED7-4E10-90E9-A5FF991AFF01}.

The **IVdsServiceIscsi** methods are specified in section 3.3.5.2.8. No methods with opnums 5, 6, 7, and 9 are listed because the VDS Protocol does not use them.

Methods in RPC Opnum Order

Method	Description
GetInitiatorName	Returns the iSCSI name of the local initiator service. Opnum: 3
QueryInitiatorAdapters	Returns an object that enumerates the iSCSI initiator adapters of the initiator. Opnum: 4
Opnum05NotUsedOnWire	Reserved for local use. Opnum: 5
Opnum06NotUsedOnWire	Reserved for local use. Opnum: 6
Opnum07NotUsedOnWire	Reserved for local use. Opnum: 7
SetInitiatorSharedSecret	Sets the initiator CHAP shared secret that is used for mutual CHAP authentication, when the initiator authenticates the target. Opnum: 8
Opnum09NotUsedOnWire	Reserved for local use. Opnum: 9

In the preceding table, the phrase "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined<35> because it does not affect interoperability.

All methods MUST NOT throw exceptions.

3.1.6 HBA Port Object Interfaces

This section includes the interfaces that are used to interact with HBA port objects on the server.

3.1.6.1 IVdsHbaPort Interface

The **IVdsHbaPort** interface provides methods to query and interact with HBA ports on the server.<36>

The UUID for this interface is {2ABD757F-2851-4997-9A13-47D2A885D6CA}.

The **IVdsHbaPort** methods are specified in section 3.3.5.2.9.

Methods in RPC Opnum Order

Method	Description
GetProperties	Retrieves the properties of the HBA port that is represented by the object exposing this interface and method. Opnum: 3
SetAllPathStatuses	Sets the statuses of all that originate from the HBA port to a specified status. Opnum: 4

All methods MUST NOT throw exceptions.

3.1.7 Initiator Adapter Object Interfaces

This section includes interfaces that are used to interact with iSCSI initiator adapter objects on the server.

3.1.7.1 IVdsIscsiInitiatorAdapter Interface

The **IVdsIscsiInitiatorAdapter** interface provides methods to query and interact with iSCSI initiator adapters on the server.<37>

The UUID for this interface is {B07FEDD4-1682-4440-9189-A39B55194DC5}.

The **IVdsIscsiInitiatorAdapter** methods are specified in section 3.3.5.2.10. No methods are listed with opnums 5 and 6 because the VDS Protocol does not use them.

Methods in RPC Opnum Order

Method	Description
GetProperties	Retrieves the properties of the initiator adapter that is represented by the object exposing this interface and method. Opnum: 3
QueryInitiatorPortals	Returns an object that enumerates the iSCSI initiator portals of the initiator

Method	Description
	adapter. Opnum: 4
Opnum05NotUsedOnWire	Reserved for local use. Opnum: 5
Opnum06NotUsedOnWire	Reserved for local use. Opnum: 6

In the preceding table, the phrase "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined<38> because it does not affect interoperability.

All methods MUST NOT throw exceptions.

3.1.8 Initiator Portal Object Interfaces

This section includes interfaces that are used to interact with iSCSI initiator portal objects on the server.

3.1.8.1 IVdsIscsiInitiatorPortal Interface

The **IVdsIscsiInitiatorPortal** interface provides methods to query and interact with iSCSI initiator portals on the server.<39>

The UUID for this interface is {38A0A9AB-7CC8-4693-AC07-1F28BD03C3DA}.

The **IVdsIscsiInitiatorPortal** methods are specified in section 3.3.5.2.11. No methods with opnums 5, 6, and 7 are listed because the VDS Protocol does not use them.

Methods in RPC Opnum Order

Method	Description
GetProperties	Retrieves the properties of the initiator portal that is represented by the object that exposes this interface and method. Opnum: 3
GetInitiatorAdapter	Returns the initiator adapter to which the initiator portal belongs. Opnum: 4
Opnum05NotUsedOnWire	Reserved for local use. Opnum: 5
Opnum06NotUsedOnWire	Reserved for local use. Opnum: 6
Opnum07NotUsedOnWire	Reserved for local use. Opnum: 7

In the table above, the phrase "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined<40> because it does not affect interoperability.

All methods MUST NOT throw exceptions.

3.1.9 Provider Object Interfaces

This section includes interfaces that are used to interact with provider objects on the server.

3.1.9.1 IVdsProvider Interface

Providers implement the **IVdsProvider** interface in order to support provider management.

The UUID for this interface is {10C5E575-7984-4E81-A56B-431F5F92AE42}.

The **IVdsProvider** methods are specified in section 3.3.5.2.12.

Methods in RPC Opnum Order

Method	Description
GetProperties	Retrieves the properties of the provider that is represented by the object exposing this interface and method. Opnum: 3

All methods MUST NOT throw exceptions.

3.1.9.2 IVdsSwProvider Interface

Software providers implement the **IVdsSwProvider** interface in order to support management of disk packs.

The UUID for this interface is {9AA58360-CE33-4F92-B658-ED24B14425B8}.

The **IVdsSwProvider** methods are specified in section 3.3.5.2.13.

Methods in RPC Opnum Order

Method	Description
QueryPacks	Retrieves the provider disk packs. Opnum: 3
CreatePack	Creates a disk pack. Opnum: 4

All methods MUST NOT throw exceptions.

3.1.9.3 IVdsHwProvider Interface

Hardware providers implement the **IVdsHwProvider** interface to support management of **subsystems**.

The UUID for this interface is {D99BDAAE-B13A-4178-9FDB-E27F16B4603E}.

The **IVdsHwProvider** methods are specified in section 3.3.5.2.14.

Methods in RPC Opnum Order

Method	Description
QuerySubSystems	Retrieves the subsystems that are managed by the provider. Opnum: 3

3.1.10 SubSystem Object Interfaces

This section includes interfaces that are used to interact with subsystem objects on the server.

3.1.10.1 IVdsSubSystemImportTarget Interface

The **IVdsSubSystemImportTarget** interface is implemented by a subsystem object to manage the **import targets** for the subsystem.

The UUID for the interface is {83BFB87F-43FB-4903-BAA6-127F01029EEC}.

The **IVdsSubSystemImportTarget** methods are specified in section 3.3.5.2.15.

Methods in RPC Opnum Order

Method	Description
GetImportTarget	Retrieves the import targets for the subsystem. Opnum: 3
SetImportTarget	Sets the import targets for the subsystem. Opnum: 4

3.1.11 Pack Object Interfaces

This section includes interfaces that are used to interact with disk pack objects on the server.

3.1.11.1 IVdsPack Interface

The **IVdsPack** interface is implemented by disk pack objects to support management of disk packs.<41>

The UUID for this interface is {3B69D7F5-9D94-4648-91CA-79939BA263BF}.

The **IVdsPack** methods are specified in section 3.3.5.2.16. No method with opnum 10 is listed because it is not used by this protocol.

Methods in RPC Opnum Order

Method	Description
GetProperties	Retrieves the properties of the disk pack that is represented by the object exposing this interface and method. Opnum: 3
GetProvider	Retrieves the provider to which the disk pack belongs. Opnum: 4
QueryVolumes	Retrieves the volumes of a disk pack.

Method	Description
	Opnum: 5
QueryDisks	Retrieves the disks of a disk pack. Opnum: 6
CreateVolume	Creates a volume in a disk pack. Opnum: 7
AddDisk	Initializes a disk that has no defined partitioning format and adds it to the disk pack. Opnum: 8
MigrateDisks	Migrates a set of disks from one pack to another pack. Opnum: 9
Opnum10NotUsedOnWire	Reserved for local use. Opnum: 10
RemoveMissingDisk	Removes the specified missing disk from a disk pack. Opnum: 11
Recover	Restores a disk pack to a healthy state. Opnum: 12

In the preceding table, the phrase "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined<42> because it does not affect interoperability.

All methods MUST NOT throw exceptions.

3.1.11.2 IVdsPack2 Interface

The **IVdsPack2** interface is implemented by disk pack objects to support creating volumes that are aligned to a particular byte-size boundary.<43>

The UUID for this interface is {13B50BFF-290A-47DD-8558-B7C58DB1A71A}.

The **IVdsPack2** methods are specified in section 3.3.5.2.17.

Methods in RPC Opnum Order

Method	Description
CreateVolume2	Creates a volume in a disk pack with an optional alignment parameter. Opnum: 3

All methods MUST NOT throw exceptions.

3.1.12 Disk Object Interfaces

This section includes interfaces that are used to interact with disk objects on the server.

3.1.12.1 IVdsDisk Interface

The **IVdsDisk** interface is implemented by disk objects in order to support disk management.<44>

The UUID for this interface is {07E5C822-F00C-47A1-8FCE-B244DA56FD06}.

The **IVdsDisk** methods are specified in section 3.3.5.2.18. No methods are listed for opnums 8 and 9 because the VDS Protocol does not use them.

Methods in RPC Opnum Order

Method	Description
GetProperties	Retrieves the properties of the disk that is represented by the object exposing this interface and method. Opnum: 3
GetPack	Retrieves the disk pack to which the disk belongs. Opnum: 4
GetIdentificationData	Retrieves information that uniquely identifies a disk. Opnum: 5
QueryExtents	Enumerates the extents of a disk. Opnum: 6
ConvertStyle	Converts the partitioning format of a disk. Opnum: 7
SetFlags	Sets the read-only flag of a disk. Opnum: 8
ClearFlags	Clears the read-only flag of a disk. Opnum: 9

All methods MUST NOT throw exceptions.

3.1.12.2 IVdsDisk2 Interface

The **IVdsDisk2** interface is implemented by disk objects in order to support bringing disks online and offline.<45>

The UUID for this interface is {40F73C8B-687D-4A13-8D96-3D7F2E683936}.

The **IVdsDisk2** methods are specified in section 3.3.5.2.19.

Methods in RPC Opnum Order

Method	Description
SetSANMode	Sets the SAN mode of a disk to either offline (**read-only** mode) or online (read/write mode). Opnum: 3

All methods MUST NOT throw exceptions.

3.1.12.3 IVdsAdvancedDisk Interface

The **IVdsAdvancedDisk** interface is implemented by disk objects in order to support advanced disk management.

The UUID for this interface is {6E6F6B40-977C-4069-BDDD-AC710059F8C0}.

The **IVdsAdvancedDisk** methods are specified in section 3.3.5.2.20.

Methods in RPC Opnum Order

Method	Description
GetPartitionProperties	Retrieves the properties of a partition on the disk at a specified byte offset. Opnum: 3
QueryPartitions	Enumerates a disk's partitions. Opnum: 4
CreatePartition	Creates a partition on a disk at a specified byte offset. Opnum: 5
DeletePartition	Deletes a partition from the disk at a specified byte offset. Opnum: 6
ChangeAttributes	Changes the attributes of the partition at byte offset ullOffset on the disk. Opnum: 7
AssignDriveLetter	Assigns a drive letter to an existing OEM, ESP, or unknown partition. Opnum: 8
DeleteDriveLetter	Deletes a drive letter that is assigned to an OEM, ESP, or unknown partition. Opnum: 9
GetDriveLetter	Retrieves the drive letter of a partition on the disk at a specified byte offset. Opnum: 10
FormatPartition	Formats an existing OEM, ESP, or unknown partition. Opnum: 11
Clean	Cleans a disk. Opnum: 12

All methods MUST NOT throw exceptions.

3.1.12.4 IVdsAdvancedDisk2 Interface

The **IVdsAdvancedDisk2** interface is implemented by disk objects in order to support changing partition types.<46>

The UUID for this interface is {9723F420-9355-42DE-AB66-E31BB15BEEAC}.

The **IVdsAdvancedDisk2** methods are specified in section 3.3.5.2.21.

Methods in RPC Opnum Order

Method	Description
ChangePartitionType	Changes the partition type on the disk at a specified byte offset. Opnum: 3

All methods MUST NOT throw exceptions.

3.1.12.5 IVdsCreatePartitionEx Interface

The **IVdsCreatePartitionEx** interface is implemented by the disk object in order to support creating partitions that are aligned to a particular byte size boundary.

The UUID for this interface is {9882F547-CFC3-420B-9750-00DFBEC50662}.

The **IVdsCreatePartitionEx** methods are specified in section 3.3.5.2.22.

Methods in RPC Opnum Order

Method	Description
CreatePartitionEx	Creates a partition on a disk at a specified byte offset, with an optional alignment parameter. Opnum: 3

All methods MUST NOT throw exceptions.

3.1.12.6 IVdsDiskPartitionMF Interface

The **IVdsDiskPartitionMF** interface is implemented by disk objects in order to support file system management on partitions.<47>

The UUID for this interface is {538684E0-BA3D-4BC0-ACA9-164AFF85C2A9}.

The **IVdsDiskPartitionMF** methods are specified in section 3.3.5.2.23.

Methods in RPC Opnum Order

Method	Description
GetPartitionFileSystemProperties	Returns property details about the file system on a partition on the disk at a specified byte offset. Opnum: 3
GetPartitionFileSystemTypeName	Retrieves the name of the file system on a partition on a disk at a specified byte offset. Opnum: 4
QueryPartitionFileSystemFormatSupport	Retrieves the properties of the file systems that are supported for formatting a partition on the disk at a specified byte offset. Opnum: 5
FormatPartitionEx	Formats an existing OEM, ESP, or unknown partition. Opnum: 6

All methods MUST NOT throw exceptions.

3.1.12.7 IVdsRemovable Interface

The **IVdsRemovable** interface is implemented by disk objects in order to support management of removable media.

The UUID for this interface is {0316560B-5DB4-4ED9-BBB5-213436DDC0D9}.

The **IVdsRemovable** methods are specified in section 3.3.5.2.24.

Methods in RPC Opnum Order

Method	Description
QueryMedia	Identifies the media in the drive. This method has no parameters. Opnum: 3
Eject	Ejects the media in the drive. This method has no parameters. Opnum: 4

All methods MUST NOT throw exceptions.

3.1.13 Volume Object Interfaces

This section includes interfaces that are used to interact with volume objects on the server.

3.1.13.1 IVdsVolume Interface

The **IVdsVolume** interface provides methods to manage volumes.

The UUID for this interface is {88306BB2-E71F-478C-86A2-79DA200A0F11}.

The **IVdsVolume** methods are specified in section 3.3.5.2.25.

Methods in RPC Opnum Order

Method	Description
GetProperties	Retrieves the properties of the volume that is represented by the object exposing this interface and method. Opnum: 3
GetPack	Retrieves the disk pack to which the volume belongs. Opnum: 4
QueryPlexes	Enumerates the plexes of a volume. Opnum: 5
Extend	Expands the size of the current volume by adding disk extents to each member of each plex. Opnum: 6
Shrink	Reduces the size of the volume and all plexes, and returns the released extents to free space.

Method	Description
	Opnum: 7
AddPlex	Adds a volume as a plex to the current volume. Opnum: 8
BreakPlex	Removes a specified plex from the current volume. Opnum: 9
RemovePlex	Removes a specified plex from a volume. The last plex of a volume cannot be removed. Opnum: 10
Delete	Deletes all plexes in a volume. Opnum: 11
SetFlags	Assigns flags to a volume. Opnum: 12
ClearFlags	Clears flags from a volume. Opnum: 13

3.1.13.2 IVdsVolumeMF Interface

The **IVdsVolumeMF** interface is implemented by volume objects in order to support file system management.

The UUID for this interface is {EE2D5DED-6236-4169-931D-B9778CE03DC6}.

The **IVdsVolumeMF** methods are specified in section 3.3.5.2.26.

Methods in RPC Opnum Order

Method	Description
GetFileSystemProperties	Returns property details about the file system on the current volume. Opnum: 3
Format	Formats a file system on the current volume. Opnum: 4
AddAccessPath	Adds an access path to the current volume. Opnum: 5
QueryAccessPaths	Returns a list of access paths and a drive letter as a single case-insensitive Unicode character, if one exists, for the current volume. Opnum: 6
QueryReparsePoints	Returns all reparse points for the current volume. Opnum: 7
DeleteAccessPath	Removes the access path from the current volume. Opnum: 8
Mount	Mounts a volume.

Method	Description
	Opnum: 9
Dismount	Dismounts a mounted volume. Opnum: 10
SetFileSystemFlags	Sets the file system flags. Opnum: 11
ClearFileSystemFlags	Clears the file system flags. Opnum: 12

All methods MUST NOT throw exceptions.

3.1.13.3 IVdsVolumeMF2 Interface

The **IVdsVolumeMF2** interface is implemented by volume objects in order to support additional file system management functionality.<48>

The UUID for this interface is {4DBCEE9A-6343-4651-B85F-5E75D74D983C}.

The **IVdsVolumeMF2** methods are specified in section 3.3.5.2.27.

Methods in RPC Opnum Order

Method	Description
GetFileSystemTypeName	Retrieves the name of the file system on a volume. Opnum: 3
QueryFileSystemFormatSupport	Retrieves the properties of the file systems that are supported for formatting a volume. Opnum: 4
FormatEx	Formats a file system on a volume. Opnum: 5

All methods MUST NOT throw exceptions.

3.1.13.4 IVdsVolumeShrink Interface

The **IVdsVolumeShrink** interface is implemented by the volume objects in order to support volume shrinking.<49>

The UUID for this interface is {D68168C9-82A2-4F85-B6E9-74707C49A58F}.

The **IVdsVolumeShrink** methods are specified in section 3.3.5.2.28.

Methods in RPC Opnum Order

Method	Description
QueryMaxReclaimableBytes	Retrieves the maximum number of bytes that can be reclaimed from the current volume.

Method	Description
	Opnum: 3
Shrink	Shrinks the volume and all plexes and returns the released extents. Opnum: 4

All methods MUST NOT throw exceptions.

3.1.13.5 IVdsVolumeOnline Interface

The **IVdsVolumeOnline** interface is implemented by the volume objects in order to support bringing single volumes online.<50>

The UUID for this interface is {1BE2275A-B315-4f70-9E44-879B3A2A53F2}.

The **IVdsVolumeOnline** methods are specified in section 3.3.5.2.29.

Methods in RPC Opnum Order

Method	Description
Online	Brings the volume online. Opnum: 3

All methods MUST NOT throw exceptions.

3.1.14 Volume Plex Object Interfaces

This section includes interfaces that are used to interact with volume plex objects on the server.

3.1.14.1 IVdsVolumePlex Interface

The **IVdsVolumePlex** interface is implemented by volume plex objects in order to support volume plex management.

The UUID for this interface is {4DAA0135-E1D1-40f1-AAA5-3CC1E53221C3}.

The **IVdsVolumePlex** methods are specified in section 3.3.5.2.30.

Methods in RPC Opnum Order

Method	Description
GetProperties	Retrieves the properties of the volume plex that are represented by the object exposing this interface and method. Opnum: 3
GetVolume	Retrieves the volume that the volume plex belongs to. Opnum: 4
QueryExtents	Returns all extents for the current plex. Opnum: 5
Repair	Repairs a fault-tolerant volume plex by moving defective members to good disks.

Method	Description
	Opnum: 6

All methods MUST NOT throw exceptions.

3.2 Client Details

3.2.1 Abstract Data Model

The client MUST maintain the following information for use in queries and commands to the server.

3.2.1.1 Notification Callback Objects

Clients can register callback objects in order to receive VDS event notifications from the server. (For more information and for an example of how clients can do this, see section 4.2.)

For each client notification callback object that is registered with the server, the client MUST maintain the following information:

Cookie: A unique 32-bit value that identifies the callback and that is maintained until the callback object is unregistered.

- The cookie is assigned by the server and returned to the client so that the client can use it to later unregister the callback.
- The client MUST NOT change the cookie.
- When the client unregisters a callback, it MUST use the cookie that the server gave to it when it originally registered the callback.

3.2.2 Timers

The VDS Protocol requires no timers.

3.2.3 Initialization

A client initializes by creating an RPC binding handle to the **IVdsServiceLoader** interface. For more information on how to get a client-side RPC binding handle for an **IVdsServiceLoader** interface, see [MS-DCOM] section 3.1.4.

After the client obtains the **IVdsServiceLoader** interface, the client MUST invoke **IVdsServiceLoader::LoadService** on the interface to retrieve the **IVdsService** interface.

After the client obtains the **IVdsService** interface, the client MUST invoke IUnknown::QueryInterface on the interface to retrieve the **IVdsServiceInitialization** interface.

After the client obtains the **IVdsServiceInitialization** interface, the client MUST invoke the **IVdsServiceInitialization::Initialize** method on the interface before invoking any other method.

After the client calls **IVdsServiceInitialization::Initialize**, the client MUST do one of the following before it invokes any other methods:

- **IVdsService::WaitForServiceReady** and wait for it to return with a success code.
- Invoke **IVdsService::IsServiceReady** in a loop until this method returns a success code.

3.2.4 Message Processing Events and Sequencing Rules

3.2.4.1 Processing Server Replies to Method Calls

After the client receives a reply from the server in response to a method call, the client MUST validate the return code. Return codes from all method calls are HRESULTs. If the HRESULT indicates success, the client can assume that any output parameters are present and valid.

Certain calls must 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 described in section 1.3.1.2. The client MUST retain the output parameters from method A until method B is called.

The client MUST release any DCOM interfaces that the server returns when the client no longer needs them.

3.2.4.2 Processing Notifications Sent from the Server to the Client

The client MAY choose to implement the **IVdsAdviseSink** interface in order to receive notification from the server of changes to the storage objects on the server. Notifications are sent to the client for creating, deleting, and modifying storage objects. The client MAY choose to take other action based on these notifications. The client MAY also choose to ignore notifications from the server.

Notifications that are related to storage object modification indicate a state change, such as when a disk status changes from VDS_DS_ONLINE to VDS_DS_FAILED, or when a volume length changes because of a call to IVdsVolume::Extend.

3.2.4.3 IVdsAdviseSink Methods

3.2.4.3.1 IVdsAdviseSink::OnNotify (Opnum 3)

The **OnNotify** method passes notifications from VDS to applications.

```
HRESULT OnNotify(
  [in, range(1,100)] long lNumberOfNotifications,
  [in, size_is(lNumberOfNotifications)]
    VDS_NOTIFICATION* pNotificationArray
);
```

lNumberOfNotifications: The number of notifications that are specified in *pNotificationArray*. This parameter MUST be a value from 1 through 100.

pNotificationArray: An array of **VDS_NOTIFICATION** structures.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

3.2.5 Timer Events

The VDS Protocol does not use any timer events.

3.2.6 Other Local Events

The VDS Protocol has no other local events that require special processing on the client.

3.3 Server Details

3.3.1 Abstract Data Model

The server MUST maintain the following information to use in responding to client queries and commands. Unless otherwise specified, zero indicates success.

The VDS Protocol uses HRESULTs, which are vendor-extensible. Vendors are free to choose their own values for this field; however, the C bit (0x20000000) must be set as specified in [MS-ERREF], indicating it is a customer code.

3.3.1.1 Service Object

The service object exposes DCOM interfaces for retrieving and interacting with all storage management objects. The service object:

- MUST implement **IVdsService** and **IVdsServiceInitialization** interfaces.
- MAY implement **IVdsServiceUninstallDisk**, **IVdsServiceHba**, and **IVdsServiceIscsi** interfaces.
- MUST maintain a value that indicates the ready state of the service. The service-ready state MAY be "not ready", "ready", or "failed". When an object is created, this value MUST be "not ready". When the service is finished initializing, this value MUST be changed to "ready". After the value is "ready", it MUST NOT change.

3.3.1.2 Storage Management Objects

The server maintains a list of the following VDS objects:

- HBA Port
  - The server MAY maintain one object for each HBA port on the system.
  - Each HBA port object exposes DCOM interfaces for querying information from an HBA port on the system.
    - Each HBA port object MUST implement the **IVdsHbaPort** interface.
- Initiator Adapter
  - The server MAY maintain one object for each initiator adapter on the system.
  - Each initiator adapter object exposes DCOM interfaces for querying information from an iSCSI initiator adapter on the system.
    - Each initiator adapter object MUST implement the **IVdsIscsiInitiatorAdapter** interface.
- Initiator Portal
  - The server MAY maintain one object for each initiator portal on the system.
  - Each initiator portal object exposes DCOM interfaces for querying information from an iSCSI initiator portal on the system.

        - Each initiator portal object MUST implement the **IVdsIscsiInitiatorPortal** interface.
    - Each initiator portal object MUST maintain a pointer to the initiator adapter object to which it belongs.
- Software Provider
    - The server MUST maintain one object for each provider on the system.
        - The basic provider is for managing **basic disks**.
        - The dynamic provider is for managing dynamic disks.
    - Each software provider object exposes DCOM interfaces for managing storage objects (packs, volumes, plexes, and disks) on the system.
        - Each software provider object MUST implement the **IVdsProvider** and **IVdsSwProvider** interfaces.
- Hardware Provider
    - The server MUST maintain one object for each hardware provider on the system.
    - Each hardware provider object exposes DCOM interfaces for managing subsystems.
    - Each hardware provider object MUST implement the **IVdsProvider** and **IVdsHwProvider** interfaces.
- Subsystem
    - The server MAY maintain one object for each subsystem on the system.
    - Each subsystem object exposes DCOM interfaces for managing the subsystem.
    - Each subsystem object MUST implement the **IVdsSubSystemImportTarget** interface.
- Pack
    - The server MUST maintain one object for each pack on the system.
    - Each pack object exposes DCOM interfaces for managing a logical group of disks and the volumes that they contain.
        - Each pack object MUST implement the **IVdsPack** interface.
        - Each pack object MAY implement the **IVdsPack2** interface.
    - Each pack object MUST maintain a pointer to the software provider object to which it belongs.
- Disk
    - The server MUST maintain for each disk on the system.
    - Each disk object exposes DCOM interfaces for managing a disk, which can include physical **hard disks**, removable disk units, optical drive units, and the LUNs that are unmasked to the system.
        - Each disk object MUST implement the **IVdsDisk** and **IVdsAdvancedDisk** interfaces.

        - If the disk is removable, the disk object MUST implement the **IVdsRemovable** interface; otherwise, it MUST NOT implement the **IVdsRemovable** interface.
        - If the disk is a removable drive with no media, the disk object MUST set its status to VDS_DS_NO_MEDIA and the values for **ulBytesPerSector**, **ulSectorsPerTrack**, **ulTracksPerCylinder**, and **ullSize** to zero.
        - Each disk object MAY implement the **IVdsDisk2**, **IVdsAdvancedDisk2**, **IVdsCreatePartitionEx**, and **IVdsDiskPartitionMF** interfaces.
    - Each disk object—if the disk is basic or dynamic—MUST maintain a pointer to the pack object to which it belongs.
- Volume
    - The server MUST maintain for each volume on the system.
    - Each volume object exposes DCOM interfaces for managing a volume, which is a logical unit of storage that exists over **regions** of one or more disks that belong to the same pack.
        - Each volume object MUST implement the **IVdsVolume**, **IVdsVolumeMF**, **IVdsVolumeMF2**, **IVdsVolumeShrink**, and **IVdsVolumeOnline** interfaces.
    - Each volume object MUST maintain a pointer to the pack object to which it belongs.
    - Removable media drives MUST contain one, and only one, volume, and the volume MUST be associated with the drive. If there is no medium in the drive, the status of the volume MUST be set to VDS_VS_NO_MEDIA and **ullsize** MUST be set to zero.<51>
- Volume Plex
    - The server MUST maintain for each volume plex on the system.
    - Each volume plex object exposes DCOM interfaces for managing a volume plex, which represents a complete copy of the data that is stored on a **mirrored volume**.
        - Each volume plex object MUST implement the **IVdsVolumePlex** interface.
        - The volume object on a removable media drive MUST contain one, and only one, volume plex, and the volume plex MUST be associated with the drive. If there is no medium in the drive, the status of the volume plex MUST be set to VDS_VPS_NO_MEDIA and **ullsize** MUST be set to zero.
    - Each volume plex object MUST maintain a pointer to the volume object to which it belongs.

Each VDS object MUST maintain the following information:

VDS Object Identifier: a unique identifier of type **VDS_OBJECT_ID**.

- The server MAY generate these identifiers at run time.
- The server MUST NOT assign two objects to the same identifier.
- The server MUST NOT change the identifier for the entire duration of a **VDS session** or until the object is removed from the list. A VDS session is defined to be from the point at which a client receives a pointer to the service object, to the point at which the client releases all references to it.

- The server MAY report different identifiers for the same object to a different VDS session—whether from the same or from different clients.
- The server SHOULD store these identifiers in the table that is associated with their respective objects in order to facilitate ID-based object retrieval.

Object Type: a value of type **VDS_OBJECT_TYPE**, which indicates the type of device that the object represents.

- When a VDS object is created, the server MUST assign its corresponding object type:
  - HBA port: VDS_OT_HBAPORT
  - Initiator adapter: VDS_OT_INIT_ADAPTER
  - Initiator portal: VDS_OT_INIT_PORTAL
  - Provider: VDS_OT_PROVIDER
  - Pack: VDS_OT_PACK
  - Disk: VDS_OT_DISK
  - Volume: VDS_OT_VOLUME
  - Volume plex: VDS_OT_VOLUME_PLEX
- The server MUST NOT ever change the object type after it is assigned.

The list of objects that the server maintains is populated when the service initializes, and is destroyed when the service shuts down. The objects can be used by more than one VDS session at a time. Objects can be added or removed from the list as a result of client requests or events that the operating system triggers, such as when a disk is no longer being reported by its bus, or when the disk's bus reports a new disk.

If objects are removed from the list while a client still has references to them, the server MUST return a value of VDS_E_OBJECT_DELETED (HRESULT of 0x8004240bL) whenever the client attempts to access the object interface methods.

3.3.1.3 Enumeration of Objects

All VDS objects that are listed in Storage Management Objects—except for the service object—are returned by means of enumeration objects. For an example of how these objects are created and used, see section 4.3.

When the client calls a method to request an enumeration, the server MUST create an enumeration object that implements the **IEnumVdsObject** interface and MUST return the interface pointer to the client to allow it to enumerate through the requested objects. The server MUST maintain this object until the client releases all references to the interface. For each enumeration object, the server MUST maintain the following information:

Objects Being Enumerated: A list of pointers to the VDS objects being enumerated.

- When the enumeration object is created, the server MUST populate this list with the objects to return, dictated by the particular specification of the method that the client calls.
- The server MUST NOT list the same object more than once.

- After the list is populated, the server MUST NOT reorder the entries in the list.
- If a new VDS object is added to the server, the server MUST NOT add the object to the list of objects being enumerated.
- If a VDS object is removed from the server, the server MUST NOT remove the object from the list of objects being enumerated. If the client later accesses the removed object, the server MUST return VDS_E_OBJECT_DELETED whenever the client attempts to access the object interface methods.

Index: A value that keeps track of which object to return next to the client, when the client requests more objects from the enumeration.

- When the enumeration object is created, this value MUST be initialized to the index of the first VDS object (whether this is 0, 1, or any other value is an implementation detail) in the list of objects being enumerated.
- If the client requests a certain number of objects from the enumeration by means of the **IEnumVdsObject::Next (Opnum 3)** method, the server MUST return the requested number of pointers to the objects in the list, starting at the current index value. However, if the server reaches the end of the list, the server MUST return the remaining pointers to the objects in the list, and indicate the actual number of objects that are returned to the client and the return code of S_FALSE. The server MUST also increment the index by the number of objects that are returned to the client.
- If the client requests to skip a certain number of objects in the enumeration by means of the **IEnumVdsObject::Skip (Opnum 4)** method, the server MUST increment the index by that number.
- If the index goes past the end of the list, all subsequent requests for more objects from the enumeration will return zero pointers and a return code of S_FALSE until the enumeration is reset.
- If the client calls the **IEnumVdsObject::Reset (Opnum 5)** method, the server MUST set the index back to the first object in the list.

3.3.1.4 Notification Callback Objects

Clients can register callback objects in order to receive VDS event notifications from the server. (For more information and for an example of how clients can do this, see section 4.2.)

For each client notification callback object that is registered with the server, the server MUST maintain the following information in its list of callback objects:

Cookie: A unique 32-bit value that identifies the callback and that is maintained until the callback object is unregistered.

- The cookie is assigned by the server and returned to the client so that the client can use it to later unregister the callback.
- The server MUST NOT change the identifier and MUST NOT assign it to another callback object until the original callback object is unregistered.

Callback Object Interface: A pointer to the **IVdsAdviseSink** Interface that is implemented by the callback object that is used to receive notifications from the server.

- Whenever a notification must be sent to the client, the server MUST call the **IVdsAdviseSink::OnNotify (Opnum 3)** method of the object in order to notify it of the event.

3.3.1.5 Asynchronous Tasks

Certain tasks in VDS may be long-running. The methods that trigger these tasks are asynchronous and have an **IVdsAsync** interface pointer as an output parameter. When the client calls a method that initiates these tasks, the server MUST create an async object that implements the **IVdsAsync** interface and returns the interface pointer to the client in order for it to monitor the task status. (For more information and for examples of how async objects can be used, see section 4.5.) The server MUST maintain this object until the client releases all references to the interface.

For each async object, the server MUST maintain the following information:

Async Output Type: A value of type **VDS_ASYNC_OUTPUT_TYPE** that indicates the type of task that the async object is monitoring.

- The server assigns this value when the object is created, and the server MUST NOT change it.

Percent Completed: An integer from 0 through 100, which indicates the percentage of progress for the task being completed.

- This value MUST be initialized to 0 when the object is created.
- If the task can be subdivided into meaningful progress milestones, the server SHOULD update this value after each milestone is passed.
- The value SHOULD always monotonically increase whenever the value is updated unless the task fails, in which case, the server MAY set the value to 0.
- If the task is successfully completed, the server MUST set the value to 100.
- When the client calls the **IVdsAsync::QueryStatus (Opnum 5)** method, the server MUST return this value in the value that the *pulPercentCompleted* output parameter references.

Signal State: A Boolean value that indicates whether the task is still in progress (FALSE); or if the task has finished, whether it finished successfully or unsuccessfully (TRUE).

- This value MUST be initialized to FALSE when the object is created.
- The server MUST change the signal state to TRUE when the task fails or when the task successfully completes.
- If the signal state is TRUE, the server MUST NOT change the signal state back to FALSE.
- If the signal state is FALSE and the client calls the **IVdsAsync::Wait (Opnum 4)** method, the server MUST block the call until the signal state is changed to TRUE, at which point the server MUST unblock the call and return the results of the task.
- If the signal state is TRUE and the client calls the **IVdsAsync::Wait (Opnum 4)** method, the server MUST return the results of the task immediately.

Return Code: An HRESULT value that indicates the final result of the task after it completes. HRESULT MUST return zero on success. If a nonzero value is returned and that value is not defined in this document, it is a generic failure.

- The server MUST set this value when the task fails or when the task successfully completes.

Task-Specific Return Values: Certain tasks MAY need to return information or pointers to objects when they complete.

- If a task (determined by **VDS_ASYNC_OUTPUT_TYPE**) returns such values, the server MUST return these values to the client by means of the **VDS_ASYNC_OUTPUT** structure that the **IVdsAsync::Wait (Opnum 4)** method returns after the task is successfully completed.

3.3.2 Timers

The VDS Protocol requires no timers.

3.3.3 Initialization

During initialization of the VDS Protocol, the service MUST start enumerating storage objects on the system and assign unique VDS object IDs to these objects, as specified in section 3.3.1.2. If service initialization has not started when the client calls the **IVdsServiceInitialization::Initialize (Opnum 3)** method, the service MUST start initializing.

3.3.3.1 Storage Management Objects

The server creates a service object and returns its interface pointer to the client that is requesting the service. The server initializes an empty list of storage management objects and populates it with provider objects that correspond to the installed providers on the system (the basic and dynamic providers). The server MUST assign each provider object a unique **VDS_OBJECT_ID**.

The server populates the list of storage management objects on the system. For more details about how each disk object is added for the basic provider, which also populates the associated pack and volume objects, see section 3.3.5.1.3. For more details about how each pack object is added for the dynamic provider, see section 3.3.5.1.1. For more details about how each disk object is added, see section 3.3.5.1.3. For more details about how each volume is added, which also populates the associated volume plex object, see section 3.3.5.1.5.

The server also queries for the HBA ports that are discoverable by using the HBA API, as well as the iSCSI initiator adapters and iSCSI initiator portals that are discoverable by using the iSCSI initiator, if they are available on the system. The service object MUST create corresponding HBA port, initiator adapter, and initiator portal objects and assign each of these objects a unique **VDS_OBJECT_ID**. For an initiator portal object, the server MUST set its initiator adapter pointer to the initiator adapter object that corresponds to the initiator adapter that contains the initiator portal. The server MUST add these objects to the list of storage management objects.

After initialization is complete, the server MUST set the service object's service-ready state to "ready". If initialization fails, the server MUST set the service object's service-ready state to "failed".

3.3.3.2 Notification Callback Objects

The server initializes an empty list of callback objects.

3.3.4 Higher-Layer Triggered Events

No higher-layer events are processed.

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Sequencing Rules

3.3.5.1.1 Adding Pack Objects for Dynamic Providers

The server MUST maintain a list of detected dynamic disk packs. When the server discovers a new pack (either during initialization or when a new pack arrives after initialization), it MUST create a corresponding pack object and MUST assign it a unique **VDS_OBJECT_ID**.

The server MUST set the pack object's provider pointer to the provider object that corresponds to the dynamic provider. The server MUST add the pack object to the list of storage management objects. For each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_PACK.
- Pack member is a **VDS_PACK_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_PACK_ARRIVE.
  - packId is the **VDS_OBJECT_ID** of the pack object that was added.

3.3.5.1.2 Removing Pack Objects for Dynamic Providers

The server MUST maintain a list of detected dynamic disk packs. When the server discovers that a pack was removed, it MUST remove the corresponding pack object from the list of storage management objects. For each callback object that is registered in the list of callback objects, the server MUST call the **IVdsAdviseSink::OnNotify (Opnum 3)** method for the callback object with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member set to VDS_NTT_PACK.
- Pack member set to a **VDS_PACK_NOTIFICATION** that has the following attributes:
  - ulEvent set to VDS_NF_PACK_DEPART.
  - packId set to the **VDS_OBJECT_ID** of the pack object that was removed.

3.3.5.1.3 Adding Disk Objects

The server MUST maintain a list of detected disks. When the server discovers a new disk (either during initialization or when a new disk arrives after initialization), it checks whether it is a basic disk, a dynamic disk, or unallocated (neither).

Basic Disk: If the disk is a basic disk, the server MUST first create a pack object and assign it a unique **VDS_OBJECT_ID**. The server MUST set the provider pointer of the pack object to the provider object that corresponds to the basic provider. The server MUST add the pack object to the list of storage management objects. For each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_PACK.
- Pack member is a **VDS_PACK_NOTIFICATION** that has the following attributes:

- ulEvent is VDS_NF_PACK_ARRIVE.
- packId is the **VDS_OBJECT_ID** of the disk object that was added.

The server MUST create a corresponding disk object and MUST assign it a unique **VDS_OBJECT_ID**. The server MUST set the disk object's pack pointer to the pack object that was created. The server MUST add the disk object to the list of storage management objects.

The server then looks for all volumes that are contained on the disk. If the disk is a removable media drive, it MUST contain one (and only one) volume that is associated with the drive itself—rather than the media. If the disk is not a removable media drive, each partition on the disk that is not an **extended partition** MAY be considered a volume.

For each volume on the disk, the server MUST create a corresponding volume object and MUST assign it a unique **VDS_OBJECT_ID**. The server MUST set the volume object's pack pointer to the pack object that was created. The server MUST add the volume object to the list of storage management objects.

For each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_DISK.
- Disk member is a **VDS_DISK_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_DISK_ARRIVE.
  - diskId is the **VDS_OBJECT_ID** of the disk object that was added.

Next, if the disk is not a removable media drive, for each partition on the disk (whether or not they are considered volumes), for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_PARTITION.
- Partition member is a **VDS_PARTITION_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_PARTITION_ARRIVE.
  - diskId is the **VDS_OBJECT_ID** of the disk object that was added.
  - ullOffset is the byte offset at which the partition starts on the disk.

Finally, for each volume on the disk, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_VOLUME.
- Volume member is a **VDS_VOLUME_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_VOLUME_ARRIVE.
  - volumeId is the **VDS_OBJECT_ID** of the volume object.

Dynamic Disk: If the disk is a dynamic disk, the server MUST create a corresponding disk object and MUST assign it a unique **VDS_OBJECT_ID**. The server MUST set the disk object's pack pointer to the pack object that corresponds to the pack that the disk belongs to.

Note that for dynamic disks, pack object creation occurs separately from disk object creation. This behavior is different from basic disks, where pack objects are created when the disk object is created, because on basic providers, packs can have only one disk. For information on pack object creation for dynamic disk packs, see section 3.3.5.1.1.

The server MUST add the disk object to the list of storage management objects. Then, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_DISK.
- Disk member is a **VDS_DISK_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_DISK_ARRIVE.
  - diskId is the **VDS_OBJECT_ID** of the disk object that was added.

Unallocated Disk: If the disk is an unallocated disk, the server MUST create a corresponding disk object and MUST assign it a unique **VDS_OBJECT_ID**. The server MUST add the disk object to the list of storage management objects. Then, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_DISK.
- Disk member is a **VDS_DISK_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_DISK_ARRIVE.
  - diskId is the **VDS_OBJECT_ID** of the disk object that was added.

3.3.5.1.4 Removing Disk Objects

The server MUST maintain a list of detected disks. When the server discovers that a disk was removed, it MUST remove the corresponding disk object from the list of storage management objects. For each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_DISK.
- disk member is a **VDS_DISK_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_DISK_DEPART.
  - diskId is the **VDS_OBJECT_ID** of the disk object that was removed.

If the disk being removed is a basic disk, the pack that the disk belongs to is also removed. In this case, the server MUST remove the corresponding pack object from the list of storage management objects. Then, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_PACK.
- disk member is a **VDS_PACK_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_DISK_ARRIVE.
  - diskId is the **VDS_OBJECT_ID** of the disk object that was added.

3.3.5.1.5 Adding Volume Objects

The server MUST maintain a list of detected volumes. When the server discovers a new volume (either during initialization or when a new volume arrives after initialization), it checks whether the volume resides on a basic disk or on one or more dynamic disks.

Basic Disk: If the volume is on a basic disk, the server MUST create a corresponding volume object and MUST assign it a unique **VDS_OBJECT_ID**. The server MUST set the volume object's pack pointer to the pack object that corresponds to the pack to which the disk on which the volume is contained belongs. The server MUST add the volume object to the list of storage management objects. For each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_VOLUME.
- volume member is a **VDS_VOLUME_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_VOLUME_ARRIVE.
  - volumeId is the **VDS_OBJECT_ID** of the volume object that was added.

Dynamic Disk: If the volume is on dynamic disks, the server MUST create a corresponding volume object and MUST assign it a unique **VDS_OBJECT_ID**. The server MUST set the volume object's pack pointer to the pack object that corresponds to the pack the volume belongs to. The server MUST add the volume object to the list of storage management objects.

For each volume plex on the volume, the server MUST create a corresponding volume plex object and MUST assign it a unique **VDS_OBJECT_ID**. The server MUST set the volume plex object's volume pointer to the volume object that was created. The server MUST add the volume plex object to the list of storage management objects.

Finally, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_VOLUME.
- volume member is a **VDS_VOLUME_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_VOLUME_ARRIVE.
  - volumeId is the **VDS_OBJECT_ID** of the volume object that was added.

3.3.5.1.6 Removing Volume Objects

The server MUST maintain a list of detected volumes. When the server discovers that a volume was removed, it MUST remove the corresponding volume object from the list of storage management objects.

If the volume is a dynamic disk, the server MUST remove the volume plex objects that correspond to the volume's volume plex from the list of storage management objects.

For each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method with a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_VOLUME.
- The volume member is a **VDS_VOLUME_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_VOLUME_DEPART.
  - volumeId is the **VDS_OBJECT_ID** of the volume object that was removed.

3.3.5.1.7 Handling Asynchronous Tasks

When the client calls a method that initiates a task that returns an async object, the server MUST create an async object that implements the **IVdsAsync** interface and return the interface pointer to the client to allow it to monitor the task's status. For examples of how async objects can be used, see section 4.5.

If the task has completed successfully and the client calls the **IVdsAsync::Wait** method on the async object, the server MUST return the following task-specific return values to the client by means of the **VDS_ASYNC_OUTPUT** structure returned by the method. The return values are determined by the async output type:

- VDS_ASYNCOUT_CREATEPARTITION:
  - The byte offset at which the partition was created (returned in the cp.ullOffset member).
  - The **VDS_OBJECT_ID** of the associated volume if the partition is a volume (returned in the cp.volumeId member).
- VDS_ASYNCOUT_CREATEVOLUME:
  - The IUnknown pointer of the volume object created (returned in the cv.pVolumeUnk member).
- VDS_ASYNCOUT_BREAKVOLUMEPLEX:
  - The IUnknown pointer of the volume object that resulted when the volume plex was broken from the original volume (returned in the bvp.pVolumeUnk member).
- VDS_ASYNCOUT_SHRINKVOLUME:
  - The number of bytes reclaimed by the shrink operation (returned in the sv.ullReclaimedBytes member).

If the async output type is none of the above or if the task did not complete successfully, no data other than the return code of the operation MAY be returned. This means that if the task fails before the method call returns to the client, the method will return an error code and MAY not return the **IVdsAsync** interface.

If the task fails after the method call has returned to the client but before the task has completed, the **IVdsAsync** interface will return an error code and MAY not contain any other information.

3.3.5.2 Message Processing Details

Before processing the methods that are listed in the following sections, the server SHOULD obtain identity and authorization information about the client from the underlying DCOM or RPC runtime. The server does this in order to verify that the client has sufficient permissions to create, modify, or delete the object as appropriate. These methods SHOULD impose an authorization policy decision before performing the function. The suggested minimum requirement is that the caller has permission to create, modify, or delete the object as appropriate.<52>

If any method is called before the server returns success from either the **IVdsService::IsServiceReady (Opnum 3)** method or the **IVdsService::WaitForServiceReady (Opnum 4)** method, the VDS_E_INITIALIZED_FAILED value is returned.

If parameter validation fails, the server MUST immediately fail the operation, returning a vendor-specific error as its response to the client.

3.3.5.2.1 IEnumVdsObject Methods

3.3.5.2.1.1 IEnumVdsObject::Next (Opnum 3)

The **Next** method returns a specified number of objects in the enumeration. It begins from the current point.

```
HRESULT Next(
  [in] unsigned long celt,
  [out, size_is(celt), length_is(*pcFetched)]
    IUnknown** ppObjectArray,
  [out] unsigned long* pcFetched
);
```

celt: The number of elements to retrieve from the enumeration.

ppObjectArray: A pointer to an array of IUnknown interfaces. The size of this array must be equal to *celt*. If successfully completed, it receives an array of the IUnknown interfaces of the next objects in the enumeration; the number of elements in this array MUST be equal in size to the value of *pcFetched*. Callers MUST release each IUnknown interface that is received.

pcFetched: A pointer to a variable that, upon successful completion, receives the number of elements that are successfully received in *ppObjectArray*.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure. If S_FALSE is returned, the client must check the value that *pcFetched* references and not access more than the number of elements returned in *ppObjectArray*.

When the server receives this message, it MUST validate the following parameters:

- Verify that *ppObjectArray* is not NULL.
- Verify that *pcFetched* is not NULL.

The server MUST perform the following:

- If the number of objects from the current index to the end of the enumeration is greater than or equal to the number of objects being requested in *celt*:

- Populate the IUnknown pointers in *ppObjectArray* with the next IUnknown pointers (the amount of which *celt* specifies) in the enumeration that is starting from the current index.
- Set the value that *pcFetched* references to *celt*.
- Increment the current index by *celt*.
- Return an HRESULT that indicates failure or success.

- If the number of objects from the current index to the end of the enumeration is less than the number of objects being requested in *celt*:
  - Populate the IUnknown pointers in *ppObjectArray* with the next IUnknown pointers in the enumeration. Start from the current index to the end of the enumeration.
  - Set the value that *pcFetched* references to the number of objects that were populated in *ppObjectArray*.
  - Increment the current index by the number of objects that were populated in *ppObjectArray*.
  - Return S_FALSE (HRESULT of 0x00000001) if successful.
- If the current index is already past the list of objects in the enumeration, set the value that *pcFetched* references to 0 and return S_FALSE (HRESULT of 0x00000001) if successful.

3.3.5.2.1.2 IEnumVdsObject::Skip (Opnum 4)

The **Skip** method skips a specified number of objects in the enumeration.

```
HRESULT Skip(
  [in] unsigned long celt
);
```

celt: The number of objects to skip.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure. If S_FALSE is returned, the enumeration has ended and the client must either stop enumerating or reset the enumeration back to the beginning.

When the server receives this message, it MUST perform the following:

- If the number of objects from the current index to the end of the enumeration is greater than or equal to the number in *celt*, increment the current index by *celt* and return an HRESULT that indicates failure or success.
- If the number of objects from the current index to the end of the enumeration is less than the number of objects that *celt* requested, increment the current index by the number of objects from the current index to the end of the enumeration and return S_FALSE (HRESULT of 0x00000001) if successful.
- If the current index is already past the list of objects in the enumeration, return S_FALSE (HRESULT of 0x00000001) if successful.

3.3.5.2.1.3 IEnumVdsObject::Reset (Opnum 5)

The **Reset** method resets the enumerator to the beginning of the collection.

```
HRESULT Reset();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST set the current index to the beginning of the enumeration and return an HRESULT that indicates failure or success.

3.3.5.2.1.4 IEnumVdsObject::Clone (Opnum 6)

The **Clone** method creates a new enumeration that has the same state as the current enumeration.

```
HRESULT Clone(
  [out] IEnumVdsObject** ppEnum
);
```

ppEnum: A pointer to an **IEnumVdsObject** interface that, if successfully completed, receives the **IEnumVdsObject** interface of the cloned enumeration. Callers MUST release the interface that is received when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the parameter:

- Verify that *ppEnum* is not NULL.

The server MUST perform the following:

- Create a new enumeration object that implements the **IEnumVdsObject** interface. Then set the pointer that *ppEnum* references to the interface.
- Set the list of objects in the new enumeration object to equal the list of objects in this enumeration.
- Set the current index in the new enumeration to equal the current index in this enumeration.
- Return an HRESULT that indicates failure or success.

3.3.5.2.2 IVdsAsync Methods

3.3.5.2.2.1 IVdsAsync::Cancel (Opnum 3)

The **Cancel** method cancels the asynchronous operation. This method has no parameters.

```
HRESULT Cancel();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST attempt to cancel the asynchronous operation and return an HRESULT that indicates failure or success. If the server succeeds in canceling the operation, it MUST set the signal state of the async object to TRUE and set the return code to VDS_E_OPERATION_CANCELED.

The server MAY set the percentage completed to 0. If it is too late to cancel the operation, the server MUST return VDS_E_CANCEL_TOO_LATE and leave the signal state of the async object and percentage completed as is.

3.3.5.2.2.2 IVdsAsync::Wait (Opnum 4)

The **Wait** method blocks and returns when the asynchronous operation has either finished successfully or failed.

```
HRESULT Wait(
  [out] HRESULT* pHrResult,
  [out] VDS_ASYNC_OUTPUT* pAsyncOut
);
```

pHrResult: A pointer to a value that, if the operation is successfully completed, receives the HRESULT that the operation returns.

pAsyncOut: A pointer to a **VDS_ASYNC_OUTPUT** structure that, if the asynchronous operation is successfully completed, receives extra information about the operation, if any information exists. Multiple methods from other interfaces also return async objects. Consult the method that returned the async object to determine what extra information to return, if any. If the asynchronous operation fails, **pAsyncOut** MAY be left as-is without returning any value.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure. The HRESULT that *pHrResult* references MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure of the asynchronous operation that is associated with the **IVdsAsync** object.

When the server receives this message, it MUST validate the following parameters:

- Verify that **pHrResult** is not NULL.
- Verify that **pAsyncOut** is not NULL.

The server MUST perform the following:

- Wait for the asynchronous operation to complete.
- Set the **pHrResult** output parameter for the operation based on the return code for the asynchronous operation.
- If the asynchronous operation has successfully completed, set the **pAsyncOut** output parameter fields based on the operation type. If the asynchronous operation has failed, this parameter may be left uninitialized.
- Return an HRESULT that indicates success or failure for the **Wait** method.

3.3.5.2.2.3 IVdsAsync::QueryStatus (Opnum 5)

The **QueryStatus** method retrieves the status of the asynchronous operation.

```
HRESULT QueryStatus(
  [out] HRESULT* pHrResult,
  [out] unsigned long* pulPercentCompleted
);
```

pHrResult: A pointer to a variable that receives the HRESULT that signals the current state of the asynchronous operation. If the asynchronous operation has finished, this parameter MUST be set to the return code associated with the operation. If the asynchronous operation is still in progress, this parameter MUST be set to VDS_S_IN_PROGRESS (HRESULT of 0x0004244D).

pulPercentCompleted: A pointer to a variable that receives the completion percentage of the asynchronous operation. If the asynchronous operation is in progress, the value MUST be between 0 and 99. If the operation has finished, the value MUST be 100. If the progress of the operation cannot be estimated, the value MUST be 0.

Return Values: The method MUST return zero on success, or an implementation-specific nonzero error code on failure. The HRESULT that *pHrResult* references MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure of the asynchronous operation that is associated with the **IVdsAsync** object.

When the server receives this message, it MUST validate the following parameters:

- Verify that **pHrResult** is not NULL.
- Verify that **pulPercentCompleted** is not NULL.

The server MUST perform the following:

- Set the **pHrResult** output parameter based on the return code for the asynchronous operation.
- Set the **pulPercentCompleted** output parameter based on the completion percentage of the asynchronous operation.
- Return an HRESULT that indicates success or failure for the **QueryStatus** method.

3.3.5.2.3 IVdsServiceLoader Methods

3.3.5.2.3.1 IVdsServiceLoader::LoadService (Opnum 3)

The **LoadService** method is used by client applications to load the VDS service on a local or remote machine.

```
HRESULT LoadService(
  [in, unique, string] LPWSTR pwszMachineName,
  [out] IVdsService** ppService
);
```

pwszMachineName: A pointer to a string that contains the name of the machine on which the loader should load the VDS service. The pointer MUST be NULL if the VDS service on the local machine is to be loaded.

ppService: A pointer to the **IVdsService** interface that, if successfully completed, returns the **IVdsService** interface to the VDS service that runs on the machine represented by *pwszMachineName*.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppService* is not NULL.

The server MUST load the VDS service and MUST point *ppService* to the **IVdsService** interface for the VDS service that is loaded, and return an **HRESULT** that indicates failure or success.

3.3.5.2.4 IVdsService Methods

3.3.5.2.4.1 IVdsService::IsServiceReady (Opnum 3)

The **IsServiceReady** method determines whether a service is finished initializing. Until the service initialization completes, an application SHOULD NOT call any method other than **GetProperties**. This method has no parameters.

```
HRESULT IsServiceReady();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

3.3.5.2.4.2 IVdsService::WaitForServiceReady (Opnum 4)

The **WaitForServiceReady** method waits for VDS initialization to complete and returns the status of the VDS initialization in the HRESULT. This method has no parameters.

```
HRESULT WaitForServiceReady();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or error code VDS_E_INITIALIZED_FAILED if the service-ready state is "failed".

Upon receiving this message, if the service-ready state is or becomes "failed", the server MUST return VDS_E_INITIALIZED_FAILED (HRESULT of 0x80042401). The server must block the call until the service-ready state is "ready", after which it MUST return success (HRESULT of 0x00000000).

3.3.5.2.4.3 IVdsService::GetProperties (Opnum 5)

The **GetProperties** method retrieves the properties of the service that is represented by the object that exposes this interface and method.

```
HRESULT GetProperties(
  [out] VDS_SERVICE_PROP* pServiceProp
```

```
);
```

pServiceProp: A pointer to a **VDS_SERVICE_PROP** structure that, if the operation is successfully completed, receives the properties of the service.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *pServiceProp* is not NULL.

The server MUST populate the **VDS_SERVICE_PROP** structure that *pServiceProp* references with the properties of the server and return an HRESULT that indicates failure or success.

3.3.5.2.4.4 IVdsService::QueryProviders (Opnum 6)

The **QueryProviders** method enumerates the providers of the server.

```
HRESULT QueryProviders(
  [in] DWORD masks,
  [out] IEnumVdsObject** ppEnum
);
```

masks: The combination of any values, by using a bitwise OR operator, that the **VDS_QUERY_PROVIDER_FLAG** enumeration defines. The values that are set in the mask specify the types of providers to return.

ppEnum: A pointer to an **IEnumVdsObject** interface that, if successfully completed, receives the **IEnumVdsObject** interface of the object that contains an enumeration of provider objects on the server. Callers MUST release the interface that is received when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppEnum* is not NULL.

The server MUST point *ppEnum* to an **IEnumVdsObject** interface that contains the enumeration of provider objects in the list of cached storage management objects, as specified in section 3.3.1.3, and return an HRESULT that indicates failure or success.

3.3.5.2.4.5 IVdsService::QueryUnallocatedDisks (Opnum 8)

The **QueryUnallocatedDisks** method enumerates the unallocated disks on the server.

```
HRESULT QueryUnallocatedDisks(
  [out] IEnumVdsObject** ppEnum
);
```

ppEnum: A pointer to an **IEnumVdsObject** interface that, if the operation is successfully completed, receives the **IEnumVdsObject** interface of the object that contains an

enumeration of disk objects that correspond to unallocated disks on the server. Callers MUST release the interface that is received when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppEnum* is not NULL.

The server MUST point *ppEnum* to an **IEnumVdsObject** interface that contains the enumeration of unallocated disk objects in the list of cached storage management objects, as specified in section 3.3.1.3, and return an HRESULT indicating failure or success.

3.3.5.2.4.6 IVdsService::GetObject (Opnum 9)

The **GetObject** method retrieves an IUnknown pointer to a specified object.

```
HRESULT GetObject(
  [in] VDS_OBJECT_ID ObjectId,
  [in] VDS_OBJECT_TYPE type,
  [out] IUnknown** ppObjectUnk
);
```

ObjectId: The GUID of the desired object.

type: The object type that **VDS_OBJECT_TYPE** enumerates. All object types are valid except VDS_OT_UNKNOWN, VDS_OT_PROVIDER, VDS_OT_ASYNC, and VDS_OT_ENUM.

ppObjectUnk: A pointer to an IUnknown interface that, if the operation is successfully completed, receives an IUnknown interface of the object. Callers MUST release the interface that is received when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppObjectUnk* is not NULL.

The server MUST point *ppObjectUnk* to an IUnknown interface of the object in the list of cached storage management objects that match the VDS object identifier that *ObjectId* specifies and the object type that is specified by type. The server MUST then return an HRESULT indicating failure or success.

3.3.5.2.4.7 IVdsService::QueryDriveLetters (Opnum 10)

The **QueryDriveLetters** method enumerates the drive letters of the server.

```
HRESULT QueryDriveLetters(
  [in] WCHAR wcFirstLetter,
  [in] DWORD count,
  [out, size_is(count)] VDS_DRIVE_LETTER_PROP* pDriveLetterPropArray
);
```

wcFirstLetter: The first drive letter to query as a single uppercase or lowercase alphabetical (A-Z) Unicode character.

count: The total number of drive letters to retrieve, beginning with the letter that *wcFirstLetter* specifies. This MUST also be the number of elements in the *pDriveLetterPropArray*. It MUST NOT exceed the total number of drive letters between the letter in *wcFirstLetter* and the last possible drive letter (Z), inclusive.

pDriveLetterPropArray: An array of **VDS_DRIVE_LETTER_PROP** structures that, if the operation is successfully completed, receives the array of drive letter properties.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that *wcFirstLetter* is an uppercase or lowercase alphabetical character (A-Z) in Unicode format.
- Verify that count does not exceed the total number of drive letters between the letter in *wcFirstLetter* and the last possible drive letter (Z), inclusive.
- Verify that *pDriveLetterPropArray* is not NULL.

The server MUST populate the **VDS_DRIVE_LETTER_PROP** structure that *pDriveLetterPropArray* references with information about each drive letter that is requested. The server MUST then return an HRESULT indicating failure or success.

3.3.5.2.4.8 IVdsService::QueryFileSystemTypes (Opnum 11)

The **QueryFileSystemTypes** method returns property details for all file systems that are known to VDS.

```
HRESULT QueryFileSystemTypes(
  [out, size_is(,*plNumberOfFileSystems)]
    VDS_FILE_SYSTEM_TYPE_PROP** ppFileSystemTypeProps,
  [out] long* plNumberOfFileSystems
);
```

ppFileSystemTypeProps: A pointer to an array of **VDS_FILE_SYSTEM_TYPE_PROP** structures that, if the operation is successfully completed, receives the array of file system type properties.

plNumberOfFileSystems: A pointer to a variable that, if the operation is successfully completed, receives the total number of elements returned in *ppFileSystemTypeProps*.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that *ppFileSystemTypeProps* is not NULL.
- Verify that *plNumberOfFileSystems* is not NULL.

The server MUST point *ppFileSystemTypeProps* to an array of **VDS_FILE_SYSTEM_TYPE_PROP** structures containing information about each file system that VDS is aware of, point *plNumberOfFileSystems* to the size of the array, and return an HRESULT indicating failure or success.

3.3.5.2.4.9 IVdsService::Reenumerate (Opnum 12)

The **Reenumerate** method discovers newly added and newly removed disks and returns the status of the operation in the HRESULT.

```
HRESULT Reenumerate();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST issue a request to all buses on the machine. The request causes the buses to report any new devices, or devices no longer present, to the operating system. The server MUST also return an HRESULT indicating failure or success.

3.3.5.2.4.10 IVdsService::Refresh (Opnum 13)

The **Refresh** method refreshes the ownership and layout of disks on the server. This method has no parameters.

```
HRESULT Refresh();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST requery the list of storage devices from the operating system, refresh its list of storage management objects based on the result of the requery, and return an HRESULT indicating failure or success.

Result of requery	Action server MUST take
New pack found	Add pack to list; see section 3.3.5.1.1
New disk found	Add disk to list; see section 3.3.5.1.3
New volume found	Add volume to list; see section 3.3.5.1.5
Pack currently in list not found	Remove pack from list; see section 3.3.5.1.2
Disk currently in list not found	Remove disk from list; see section 3.3.5.1.4
Volume currently in list not found	Remove volume from list; see section 3.3.5.1.6

3.3.5.2.4.11 IVdsService::CleanupObsoleteMountPoints (Opnum 14)

The **CleanupObsoleteMountPoints** method removes any mount points that point to volumes that no longer exist. This method has no parameters.

```
HRESULT CleanupObsoleteMountPoints();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST requery the assigned mount points from the operating system, remove mount points from the operating system if they are assigned to volumes that no longer exist, and return an HRESULT indicating failure or success.

3.3.5.2.4.12 IVdsService::Advise (Opnum 15)

The **Advise** method registers a notification callback with the server. Clients pass the callback object to the server to receive notifications.

```
HRESULT Advise(
  [in] IVdsAdviseSink* pSink,
  [out] DWORD* pdwCookie
);
```

pSink: A pointer to an IVdsAdviseSink interface of the callback object to register with the server for notification of object changes.

pdwCookie: A pointer to a variable that, if the operation is successfully completed, receives a unique cookie value that the client can later use to unregister the callback object from receiving notification changes from the service. For information about how to register callback objects, see section 3.2.1.1.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that *pSink* is not NULL.
- Verify that *pdwCookie* is not NULL.

The server MUST perform the following:

- Point *pdwCookie* to a unique cookie value that is associated with the IVdsAdviseSink interface that *pSink* specifies.
- Add the IVdsAdviseSink interface that *pSink* specifies to the list of callback objects.
- Return an HRESULT indicating failure or success.

3.3.5.2.4.13 IVdsService::Unadvise (Opnum 16)

The **Unadvise** method unregisters a client from being notified by the server of changes to storage objects.

```
HRESULT Unadvise(
  [in] DWORD dwCookie
);
```

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *dwCookie* corresponds to a callback object in the list of callback objects.

The server MUST return an HRESULT indicating failure or success.

3.3.5.2.4.14 IVdsService::Reboot (Opnum 17)

The **Reboot** method restarts the computer on which the server is running. This method has no parameters.

```
HRESULT Reboot();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST send a request to the operating system to restart the computer and return an HRESULT indicating failure or success.

3.3.5.2.4.15 IVdsService::SetFlags (Opnum 18)

The **SetFlags** method assigns property flags to the server.

```
HRESULT SetFlags(
  [in] unsigned long ulFlags
);
```

ulFlags: A value from the **VDS_SERVICE_FLAG** enumeration. Only the VDS_SVF_AUTO_MOUNT_OFF flag is valid for this method.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ulFlags* does not contain any flags that the parameter specification disallows.

The server MUST attempt to set the service flags that *ulFlags* specifies and return an HRESULT indicating failure or success.

3.3.5.2.4.16 IVdsService::ClearFlags (Opnum 19)

The **ClearFlags** method clears property flags from the service.

```
HRESULT ClearFlags(
  [in] unsigned long ulFlags
);
```

ulFlags: A value from the **VDS_SERVICE_FLAG** enumeration. Only the VDS_SVF_AUTO_MOUNT_OFF flag is valid for this method.

Return Values: The method returns zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ulFlags* does not contain any flags that the parameter specification disallows.

The server MUST attempt to clear the service flags that *ulFlags* specifies and return an HRESULT indicating failure or success.

3.3.5.2.5 IVdsServiceInitialization Methods

3.3.5.2.5.1 IVdsServiceInitialization::Initialize (Opnum 3)

The **Initialize** method starts the initialization of the server.

```
HRESULT Initialize(
  [in, unique, string] WCHAR* pwszMachineName
);
```

pwszMachineName: This parameter is reserved and not used. Callers should pass in NULL.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *pwszMachineName* is NULL.

If the parameter validation fails, the server MUST immediately fail the operation, returning a vendor-specific error as its response to the client. If the parameter validation is successful, the server MUST initialize the server and return a success value (HRESULT of 0x00000000) if the operation is successful, or VDS_E_INITIALIZED_FAILED (HRESULT of 0x80042401) if the operation failed. For more information about initialization, see section 3.3.3.

3.3.5.2.6 IVdsServiceUninstallDisk Methods

3.3.5.2.6.1 IVdsServiceUninstallDisk::GetDiskIdFromLunInfo (Opnum 3)

The **GetDiskIdFromLunInfo** method retrieves the VDS object ID of a disk that corresponds to a specified LUN information structure.<53>

```
HRESULT GetDiskIdFromLunInfo(
  [in] VDS_LUN_INFORMATION* pLunInfo,
  [out] VDS_OBJECT_ID* pDiskId
);
```

pLunInfo: A pointer to a VDS_LUN_INFORMATION structure that stores the disk's LUN information.

pDiskId: A pointer to a **VDS_OBJECT_ID** structure that, if the operation is successfully completed, receives the VDS object ID of the disk object that corresponds to the LUN information that *pLunInfo* specifies.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that *pLunInfo* is not NULL.
- Verify that *pDiskId* is not NULL.

The server MUST set *pDiskId* to the VDS object identifier of the disk that matches the LUN information that *pLunInfo* specifies, and return an HRESULT that indicates failure or success.

3.3.5.2.6.2 IVdsServiceUninstallDisk::UninstallDisks (Opnum 4)

The **UninstallDisks** method uninstalls a specific set of disks when it is given a list of the VDS object IDs for the disks. All volumes that are contained wholly or partially on the disks are also uninstalled, and the obsolete mount points are removed.<54>

```
HRESULT UninstallDisks(
  [in, size_is(ulCount)] VDS_OBJECT_ID* pDiskIdArray,
  [in] unsigned long ulCount,
  [in] boolean bForce,
  [out] boolean* pbReboot,
  [out, size_is(ulCount)] HRESULT* pResults
);
```

pDiskIdArray: A pointer to an array of **VDS_OBJECT_ID** structures that store the VDS object IDs of the disks to be uninstalled.

ulCount: The number of disks that are specified in *pDiskIdArray*.

bForce: A Boolean that determines whether the volume dismount is forced.

pbReboot: A pointer to a Boolean that, if the operation is successfully completed, receives an indication whether the user must reboot the remote machine in order to complete the uninstall process.

pResults: A pointer to an array of HRESULT values that, if the operation is successfully completed, receives the HRESULTs that each disk uninstall request returned. There MUST be one HRESULT value in the array for each disk in *pDiskIdArray*. If any of the disks fail to uninstall properly, the specific error code for that failure is received in the corresponding entry in *pResults*.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that *pDiskIdArray* is not NULL.
- Verify that *pbReboot* is not NULL.
- Verify that *pResults* is not NULL.

The server MUST perform the following:

- For each VDS object identifier in the specified **pDiskIdArray**, lock and dismount all volumes that have extents on the disk. If the value of the Boolean that bForce references is specified as TRUE, continue to the next steps, even if the lock or dismount operation fails.
- For each VDS object identifier in the specified **pDiskIdArray**, take offline (if possible) and uninstall all volumes that have extents on the disk. For information on removing volumes, see section 3.3.5.1.6.
- If the volume uninstalls require a reboot to take full effect, set the value of the Boolean that **pbReboot** references to TRUE; otherwise, FALSE.
- For each VDS object identifier in the specified *pDiskIdArray*, uninstall the disk that corresponds to the identifier. Set the status of each disk operation to the corresponding value in the array that *pResults* specifies. For more information on removing disk objects, see section 3.3.5.1.4.
- If the disk uninstalls require a restart to take full effect, set the value of the Boolean that *pbReboot* references to TRUE; otherwise, FALSE.
- Clean up any obsolete drive letters and mount points for the volumes that have been uninstalled.
- Return success (HRESULT of 0x00000000) if successful.

3.3.5.2.7 IVdsServiceHba Methods

3.3.5.2.7.1 IVdsServiceHba::QueryHbaPorts (Opnum 3)

The **QueryHbaPorts** method returns an **IEnumVdsObject** enumeration object that contains a list of the HBA ports that are known to VDS on the local system.<55>

```
HRESULT QueryHbaPorts(
  [out] IEnumVdsObject** ppEnum
);
```

ppEnum: A pointer to an **IEnumVdsObject** interface that, if the operation is successfully completed, receives the **IEnumVdsObject** interface of the object that contains an enumeration of the HBA port objects on the server. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppEnum* is not NULL.

The server MUST point *ppEnum* to an **IEnumVdsObject** interface that contains the enumeration of the HBA port objects in the list of cached storage management objects, as specified in section 3.3.1.3, and return an HRESULT indicating failure or success.

3.3.5.2.8 IVdsServiceIscsi Methods

3.3.5.2.8.1 IVdsServiceIscsi::GetInitiatorName (Opnum 3)

The **GetInitiatorName** method returns the iSCSI name of the local initiator service.<56>

```
HRESULT GetInitiatorName(
  [out, string] WCHAR** ppwszIscsiName
);
```

ppwszIscsiName: A pointer that, if the operation is successfully completed, receives a null-terminated Unicode string with the iSCSI name.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppwszIscsiName* is not NULL.

The server MUST set *ppwszIscsiName* to point to a string that contains the iSCSI name of the iSCSI initiator on the system, and return an HRESULT indicating failure or success.

3.3.5.2.8.2 IVdsServiceIscsi::QueryInitiatorAdapters (Opnum 4)

The **QueryInitiatorAdapters** method returns an object that enumerates the iSCSI initiator adapters of the initiator.<57>

```
HRESULT QueryInitiatorAdapters(
  [out] IEnumVdsObject** ppEnum
);
```

ppEnum: A pointer to an **IEnumVdsObject** interface that, if the operation is successfully completed, receives the **IEnumVdsObject** interface of the object that contains an enumeration of initiator adapter objects on the server. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppEnum* is not NULL.

The server MUST point *ppEnum* to an **IEnumVdsObject** interface that contains the enumeration of iSCSI initiator adapter objects in the list of cached storage management objects, as specified in section 3.3.1.3, and return an HRESULT indicating failure or success.

3.3.5.2.8.3 IVdsServiceIscsi::SetInitiatorSharedSecret (Opnum 8)

The **SetInitiatorSharedSecret** method sets the initiator CHAP shared secret that is used for mutual CHAP authentication when the initiator authenticates the target. For more information on CHAP, see [MS-CHAP].<58>

```
HRESULT SetInitiatorSharedSecret(
  [in, unique] VDS_ISCSI_SHARED_SECRET* pInitiatorSharedSecret,
  [in] VDS_OBJECT_ID targetId
);
```

pInitiatorSharedSecret: A pointer to a VDS_ISCSI_SHARED_SECRET structure that contains the CHAP shared secret that is used for mutual CHAP authentication when the initiator authenticates the target. For more information on CHAP, see [MS-CHAP].

targetId: This parameter is reserved and not used by the protocol. Callers should pass in GUID_NULL.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that *pInitiatorSharedSecret* is not NULL.
- Verify that *targetId* is GUID_NULL.

The server MUST set the CHAP shared secret that *pInitiatorSharedSecret* specifies in the iSCSI initiator, and return an HRESULT indicating failure or success.

3.3.5.2.9 IVdsHbaPort Methods

3.3.5.2.9.1 IVdsHbaPort::GetProperties (Opnum 3)

The **GetProperties** method retrieves the properties of the HBA port that the object exposing this interface and method represents.<59>

```
HRESULT GetProperties(
  [out] VDS_HBAPORT_PROP* pHbaPortProp
);
```

pHbaPortProp: A pointer to a **VDS_HBAPORT_PROP** structure that, if the operation is successfully completed, receives the properties of the HBA port.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *pHbaPortProp* is not NULL.

The server MUST populate the **VDS_HBAPORT_PROP** structure that *pHbaPortProp* references with the properties of the HBA port, and return an HRESULT that indicates failure or success. For more information on the VDS_HBAPORT_PROP structure, see section 2.2.2.3.2.2.

3.3.5.2.9.2 IVdsHbaPort::SetAllPathStatuses (Opnum 4)

The **SetAllPathStatuses** method sets the statuses of all that originate from the HBA port to a specified status.<60>

```
HRESULT SetAllPathStatuses(
  [in] VDS_PATH_STATUS status
);
```

status: The status, as defined by **VDS_PATH_STATUS**, to assign to the paths.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

3.3.5.2.10 IVdsIscsiInitiatorAdapter Methods

3.3.5.2.10.1 IVdsIscsiInitiatorAdapter::GetProperties (Opnum 3)

The **GetProperties** method retrieves the properties of the initiator adapter that is represented by the object exposing this interface and method.<61>

```
HRESULT GetProperties(
  [out] VDS_ISCSI_INITIATOR_ADAPTER_PROP* pInitiatorAdapterProp
);
```

pInitiatorAdapterProp: A pointer to a **VDS_ISCSI_INITIATOR_ADAPTER_PROP** structure that, if the operation is successfully completed, receives the properties of the initiator adapter.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *pInitiatorAdapterProp* is not NULL.

The server MUST populate the **VDS_ISCSI_INITIATOR_ADAPTER_PROP** structure that *pInitiatorAdapterProp* references with the properties of the iSCSI initiator adapter, and return an HRESULT indicating failure or success. For information on the VDS_ISCSI_INITIATOR_ADAPTER_PROP structure, see section 2.2.2.4.1.1.

3.3.5.2.10.2 IVdsIscsiInitiatorAdapter::QueryInitiatorPortals (Opnum 4)

The **QueryInitiatorPortals** method returns an object that enumerates the iSCSI initiator portals of the initiator adapter.<62>

```
HRESULT QueryInitiatorPortals(
  [out] IEnumVdsObject** ppEnum
);
```

ppEnum: A pointer to an **IEnumVdsObject** interface that, if the operation is successfully completed, receives the **IEnumVdsObject** interface of the object containing an enumeration

of initiator portal objects in the initiator adapter. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the parameters:

- Verify that *ppEnum* is not NULL.

The server MUST point *ppEnum* to an **IEnumVdsObject** interface that contains the enumeration of iSCSI initiator portal objects in the list of cached storage management objects that have their initiator adapter pointer set to this initiator adapter object. The server MUST then return an HRESULT indicating failure or success.

3.3.5.2.11 IVdsIscsiInitiatorPortal Methods

3.3.5.2.11.1 IVdsIscsiInitiatorPortal::GetProperties (Opnum 3)

The **GetProperties** method retrieves the properties of the initiator portal that the object exposing this interface and method represents.<63>

```
HRESULT GetProperties(
  [out] VDS_ISCSI_INITIATOR_PORTAL_PROP* pInitiatorPortalProp
);
```

pInitiatorPortalProp: A pointer to a **VDS_ISCSI_INITIATOR_PORTAL_PROP** structure that, if the operation is successfully completed, receives the properties of the initiator portal.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *pInitiatorPortalProp* is not NULL.

The server MUST populate the **VDS_ISCSI_INITIATOR_PORTAL_PROP** structure that *pInitiatorPortalProp* references with the properties of the iSCSI initiator portal, and then return an HRESULT indicating failure or success. For information on the VDS_ISCSI_INITIATOR_PORTAL_PROP structure, see section 2.2.2.5.2.2.

3.3.5.2.11.2 IVdsIscsiInitiatorPortal::GetInitiatorAdapter (Opnum 4)

The **GetInitiatorAdapter** method returns the initiator adapter to the initiator portal it belongs to.<64>

```
HRESULT GetInitiatorAdapter(
  [out] IVdsIscsiInitiatorAdapter** ppInitiatorAdapter
);
```

ppInitiatorAdapter: A pointer to an **IVdsIscsiInitiatorAdapter** interface that, if the operation is successfully completed, receives the **IVdsIscsiInitiatorAdapter** interface of the initiator adapter object that the initiator portal belongs to. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppInitiatorAdapter* is not NULL.

The server MUST point *ppInitiatorAdapter* to an **IVdsIscsiInitiatorAdapter** interface of the initiator adapter object that the initiator portal object's initiator adapter pointer refers to. The server MUST then return an HRESULT indicating failure or success.

3.3.5.2.12 IVdsProvider Methods

3.3.5.2.12.1 IVdsProvider::GetProperties (Opnum 3)

The **GetProperties** method retrieves the properties of the provider that the object exposing this interface and method represents.

```
HRESULT GetProperties(
  [out] VDS_PROVIDER_PROP* pProviderProp
);
```

pProviderProp: A pointer to a **VDS_PROVIDER_PROP** structure that, if the operation is successfully completed, receives the properties of the provider.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *pProviderProp* is not NULL.

The server MUST populate the **VDS_PROVIDER_PROP** structure that *pProviderProp* references with the properties of the provider. It MUST then return an HRESULT indicating failure or success. For information on the VDS_PROVIDER_PROP structure, see section 2.2.2.6.2.1.

3.3.5.2.13 IVdsSwProvider Methods

3.3.5.2.13.1 IVdsSwProvider::QueryPacks (Opnum 3)

The **QueryPacks** method retrieves the provider disk packs.

```
HRESULT QueryPacks(
  [out] IEnumVdsObject** ppEnum
);
```

ppEnum: A pointer to an **IEnumVdsObject** interface that, if the operation is successfully completed, receives the **IEnumVdsObject** interface of the object containing an enumeration of pack objects in the provider. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppEnum* is not NULL.

The server MUST point *ppEnum* to an **IEnumVdsObject** interface that contains the enumeration of pack objects in the list of cached storage management objects that have their provider pointer set to this provider object. The server MUST then return an HRESULT indicating failure or success. For information on enumeration objects, see section 3.3.1.3.

3.3.5.2.13.2 IVdsSwProvider::CreatePack (Opnum 4)

The **CreatePack** method creates a disk pack.

```
HRESULT CreatePack(
  [out] IVdsPack** ppPack
);
```

ppPack: A pointer to an **IVdsPack** interface that, if the operation is successfully completed, receives the **IVdsPack** interface of the newly created disk pack. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppPack* is not NULL.

The server MUST perform the following:

- Create a new pack object that implements the **IVdsPack** interface and assign it a unique **VDS_OBJECT_ID**.
- Set the provider pointer of the disk pack object to this provider object.
- Add the pack object to the list of storage management objects.
- For each callback object that is registered in the list of callback objects, call the **IVdsAdviseSink::OnNotify (Opnum 3)** method of the callback object with a VDS_NOTIFICATION structure that has the following attributes:
  - **objectType** member is VDS_NTT_PACK.
  - **Pack** member is a VDS_PACK_NOTIFICATION with the following attributes:
    - **ulEvent** is VDS_NF_PACK_ARRIVE.
    - **packId** is the **VDS_OBJECT_ID** of the pack object that was added.
- Set the pointer that *ppPack* references to the **IVdsPack** interface of the pack object.
- Return an HRESULT indicating failure or success.

3.3.5.2.14 IVdsHwProvider Methods

3.3.5.2.14.1 IVdsHwProvider::QuerySubSystems (Opnum 3)

The **QuerySubSystems** method retrieves the subsystems that are managed by the provider.

```
HRESULT QuerySubSystems(
  [out] IEnumVdsObject** ppEnum
);
```

ppEnum: A pointer to an **IEnumVdsObject** interface. If the operation is successfully completed, the pointer receives the **IEnumVdsObject** interface of the object, which contains an enumeration of subsystem objects in the provider. Callers MUST release the interface when they are finished with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppEnum* is not NULL.

The server MUST point *ppEnum* to an **IEnumVdsObject** interface that contains the enumeration of subsystem objects in the list of cached storage management objects that have their provider pointer set to this provider object. The server MUST then return an **HRESULT** to indicate failure or success.

For information on enumeration objects, see section 3.3.1.3.

3.3.5.2.15 IVdsSubSystemImportTarget Methods

3.3.5.2.15.1 IVdsSubSystemImportTarget::GetImportTarget (Opnum 3)

The **GetImportTarget** method retrieves the name of the import target to associate with the LUNs being imported on the subsystem.

```
HRESULT GetImportTarget(
  [out, string] LPWSTR* ppwszIscsiName
);
```

ppwszIscsiName: A pointer to a string that contains the name of the import target of the subsystem. Callers MUST free the memory that is allocated for the string when they are finished with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppwszIscsiName* is not NULL.

The server MUST point *ppwszIscsiName* to a string that contains the name of the import target of the subsystem. The server MUST then return an **HRESULT** to indicate failure or success.

3.3.5.2.15.2 IVdsSubSystemImportTarget::SetImportTarget (Opnum 4)

The **SetImportTarget** method sets the name of the import target to associate with the LUNs being imported on the subsystem.

```
HRESULT SetImportTarget(
```

```
  [in, unique, string] LPWSTR pwszIscsiName
);
```

pwszIscsiName: A string that contains the name of the import target of the subsystem.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *pwszIscsiName* is not NULL.

The server MUST set the name of the import target of the subsystem to the string that is specified by *ppwszIscsiName*. The server MUST then return an **HRESULT** to indicate failure or success.

3.3.5.2.16 IVdsPack Methods

3.3.5.2.16.1 IVdsPack::GetProperties (Opnum 3)

The **GetProperties** method retrieves the properties of the disk pack that the object exposing this interface and method represents.

```
HRESULT GetProperties(
  [out] VDS_PACK_PROP* pPackProp
);
```

pPackProp: A pointer to a **VDS_PACK_PROP** structure that, if the operation is successfully completed, receives the properties of the pack.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *pPackProp* is not NULL.

The server MUST populate the **VDS_PACK_PROP** structure that *pPackProp* references with the properties of the pack. It MUST then return an HRESULT indicating failure or success. For information on the VDS_PACK_PROP structure, see section 2.2.2.7.2.1.

3.3.5.2.16.2 IVdsPack::GetProvider (Opnum 4)

The **GetProvider** method retrieves the provider that the disk pack belongs to.

```
HRESULT GetProvider(
  [out] IVdsProvider** ppProvider
);
```

ppProvider: A pointer to an **IVdsProvider** interface that, if the operation is successfully completed, receives the **IVdsProvider** interface of the provider object that the pack belongs to. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppProvider* is not NULL.

The server MUST point *ppProvider* to an **IVdsProvider** interface of the provider object that the pack object's provider pointer refers to. The server MUST then return an HRESULT indicating failure or success.

3.3.5.2.16.3 IVdsPack::QueryVolumes (Opnum 5)

The **QueryVolumes** method retrieves the volumes of a disk pack.

```
HRESULT QueryVolumes(
  [out] IEnumVdsObject** ppEnum
);
```

ppEnum: A pointer to an **IEnumVdsObject** interface that, if the operation is successfully completed, receives the **IEnumVdsObject** interface of the object that contains an enumeration of volume objects in the pack. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppEnum* is not NULL.

The server MUST point *ppEnum* to an **IEnumVdsObject** interface that contains the enumeration of volume objects in the list of cached storage management objects that have their pack pointer set to this pack object. The server MUST then return an HRESULT indicating failure or success. For information on enumeration objects, see section 3.3.1.3.

3.3.5.2.16.4 IVdsPack::QueryDisks (Opnum 6)

The **QueryDisks** method retrieves the disks of a disk pack.

```
HRESULT QueryDisks(
  [out] IEnumVdsObject** ppEnum
);
```

ppEnum: A pointer to an **IEnumVdsObject** interface that, if the operation is successfully completed, receives the **IEnumVdsObject** interface of the object containing an enumeration of disk objects in the pack. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppEnum** is not NULL.

The server MUST point **ppEnum** to an **IEnumVdsObject** interface that contains the enumeration of disk objects in the list of cached storage management objects that have their pack pointer set to this pack object. The server MUST then return an HRESULT indicating failure or success. For information on enumeration, see section 3.3.1.3.

3.3.5.2.16.5 IVdsPack::CreateVolume (Opnum 7)

The **CreateVolume** method creates a volume in a disk pack.

```
HRESULT CreateVolume(
  [in] VDS_VOLUME_TYPE type,
  [in, size_is(lNumberOfDisks)] VDS_INPUT_DISK* pInputDiskArray,
  [in] long lNumberOfDisks,
  [in] unsigned long ulStripeSize,
  [out] IVdsAsync** ppAsync
);
```

type: A value from the **VDS_VOLUME_TYPE** enumeration that indicates the type of volume to create.

pInputDiskArray: An array of **VDS_INPUT_DISK** structures that indicate the disks on which to create the volume.<65>

lNumberOfDisks: The number of elements in **pInputDiskArray**.

ulStripeSize: The stripe size of the new volume.<66>

ppAsync: A pointer to an **IVdsAsync** interface that, if the operation is successfully completed, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface when they are done with it. If the **IVdsAsync::Wait (Opnum 4)** method is called on the interface, the interfaces returned in the **VDS_ASYNC_OUTPUT** structure MUST be released as well. For information on handling asynchronous tasks, see section 3.3.5.1.7.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **pInputDiskArray** is not NULL.
- Verify that **ppAsync** is not NULL.

The server MUST perform the following:

- Create a new async object implementing the **IVdsAsync** interface with an output type of VDS_ASYNCOUT_CREATEVOLUME and set the pointer that **ppAsync** references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Create a new volume that uses the parameters that are specified for this method.
  - If the volume creation resulted in the renumbering of existing partitions on a boot disk, the server MUST update the **boot configuration file** with the new partition numbering.

- Create a new volume object that corresponds to the new volume, implement the **IVdsVolume** interface, and assign it a unique **VDS_OBJECT_ID**.
- Set the volume object's pack pointer to this pack object.
- Create new volume plex objects that correspond to the new volume plexes, implement the IVdsVolumePlex interface, and assign it a unique **VDS_OBJECT_ID**.
- Set the volume plex object's volume pointer to this volume object.
- Add the volume plex object to the list of storage management objects.
- Add the volume object to the list of storage management objects.
- Set the task-specific return values in the async object to return the values that are associated with VDS_ASYNCOUT_CREATEVOLUME. See section 3.3.5.1.7.
- Set the return code in the async object to an HRESULT indicating failure or success.
  - If the server tried to update the boot configuration file but failed, the return code MUST be set to VDS_S_UPDATE_BOOTFILE_FAILED (HRESULT of 0x80042434).
- If the task completed successfully, set the percentage completed value in the async object to 100.
- Set the signal state in the async object to TRUE.
- For each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Pack** member is a **VDS_VOLUME_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_ARRIVE.
    - **volumeId** is the **VDS_OBJECT_ID** of the volume object that was added.

At any point in the above sequence—before the percentage completed value in the async object is 100—the server MAY update the percentage completed value.

3.3.5.2.16.6 IVdsPack::AddDisk (Opnum 8)

This method initializes a disk that has no partitioning format defined and then adds it to the disk pack. You cannot use **AddDisk** to redefine the partitioning format on a disk.<67>

```
HRESULT AddDisk(
  [in] VDS_OBJECT_ID DiskId,
  [in] VDS_PARTITION_STYLE PartitionStyle,
  [in] long bAsHotSpare
);
```

DiskId: The VDS object ID of the disk object.

PartitionStyle: A value from the **VDS_PARTITION_STYLE** enumeration that indicates the partition format.

bAsHotSpare: The VDS Protocol does not support this parameter; callers SHOULD set it to FALSE.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **DiskId** belongs to a disk object in the list of storage management objects corresponding to an unallocated disk.
- Verify that **PartitionStyle** is a valid partitioning format.

The server MUST add the disk having a **VDS_OBJECT_ID** specified by **DiskId** to this pack, initialize the partitioning format of the disk based on the value of the **PartitionStyle** parameter, set the disk object's pack pointer to this pack object, and return an HRESULT indicating failure or success.

3.3.5.2.16.7 IVdsPack::MigrateDisks (Opnum 9)

The **MigrateDisks** method migrates a set of disks from one pack to another pack.

```
HRESULT MigrateDisks(
  [in, size_is(lNumberOfDisks)] VDS_OBJECT_ID* pDiskArray,
  [in] long lNumberOfDisks,
  [in] VDS_OBJECT_ID TargetPack,
  [in] long bForce,
  [in] long bQueryOnly,
  [out, size_is(lNumberOfDisks)]
    HRESULT* pResults,
  [out] long* pbRebootNeeded
);
```

pDiskArray: A pointer to an array of VDS object IDs—one for each disk object that corresponds to the disks to migrate.

lNumberOfDisks: The number of disks specified in pDiskArray.

TargetPack: The VDS object ID of the pack object.

bForce: A Boolean that determines whether the disk migration is forced.

bQueryOnly: A Boolean that determines whether the disk migration will actually happen.

pResults: A pointer to an array of HRESULT values that, if the operation is successfully completed, receives the HRESULTs returned by each disk migration request. There MUST be one HRESULT value in the array for each disk in pDiskArray. If any of the disks fail to migrate properly, the specific error code for that failure is received in the corresponding entry in pResults.

pbRebootNeeded: A pointer to a Boolean that, if the operation is successfully completed, receives an indication of whether the user must reboot the remote machine in order to complete the migration process.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure. The HRESULTs in the array that pResults

references MUST return zero to indicate success, or must return an implementation-specific nonzero error code if the migration operation on the associated disk fails.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST validate the following parameters:

- Verify that **pDiskArray** is not NULL.
- Verify that **pResults** is not NULL.
- Verify that **pbRebootNeeded** is not NULL.

The server MUST perform the following:

- For each VDS object identifier in the specified pDiskArray, migrate the disk that corresponds to the identifier to this pack. Set the status of each migrate operation to the corresponding value in the array that pResults specifies.
- For each successfully migrated disk, set the disk object's pack pointer to this pack object.
- If the disk migrations require a restart to take full effect, set the value of the Boolean that **pbRebootNeeded** references to TRUE; otherwise, FALSE.
- Return an HRESULT indicating failure or success; also return an HRESULT for each disk that is involved in the migration by using the pResults output parameter.

3.3.5.2.16.8 IVdsPack::RemoveMissingDisk (Opnum 11)

The **IVdsPack::RemoveMissingDisk** method removes the specified missing disk from a disk pack.

```
HRESULT RemoveMissingDisk(
  [in] VDS_OBJECT_ID DiskId
);
```

DiskId: The VDS object ID of the disk object to remove.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the parameters:

- Verify that **DiskId** belongs to a disk object in the list of storage management objects that correspond to a disk that is missing. The missing disk has its pack pointer set to this pack object.

The server MUST remove the disk having a **VDS_OBJECT_ID** that is specified by **DiskId** to this pack, remove the corresponding disk object from the list of storage management objects, and return an HRESULT indicating failure or success.

3.3.5.2.16.9 IVdsPack::Recover (Opnum 12)

The **Recover** method restores a disk pack to a healthy state.

```
HRESULT Recover(
  [out] IVdsAsync** ppAsync
```

```
);
```

ppAsync: A pointer to an IVdsAsync interface that, if the operation is successfully completed, receives the IVdsAsync interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppAsync** is not NULL.

The server MUST perform the following:

- Create a new async object implementing the IVdsAsync interface with an output type of VDS_ASYNCOUT_RECOVERPACK and set the pointer **ppAsync** references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Bring online all disks in the online pack, attempt to resync any mirrored volumes, and regenerate any RAID5 volumes that are in the online pack.
- Set the return code in the async object to an HRESULT indicating failure or success.
- If the task completed successfully, set the percentage completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the above sequence—before the percentage completed value in the async object is 100—the server MAY update the percentage completed value.

3.3.5.2.17 IVdsPack2 Methods

3.3.5.2.17.1 IVdsPack2::CreateVolume2 (Opnum 3)

The **CreateVolume2** method creates a volume in a disk pack with an optional alignment parameter.

```
HRESULT CreateVolume2(
  [in] VDS_VOLUME_TYPE type,
  [in, size_is(lNumberOfDisks)] VDS_INPUT_DISK* pInputDiskArray,
  [in] long lNumberOfDisks,
  [in] unsigned long ulStripeSize,
  [in] unsigned long ulAlign,
  [out] IVdsAsync** ppAsync
);
```

type: A value from the **VDS_VOLUME_TYPE** enumeration that indicates the type of volume to create.

pInputDiskArray: An array of **VDS_INPUT_DISK** structures that indicate the disks on which to create the volume.<68>

lNumberOfDisks: The number of elements in **pInputDiskArray**.

ulStripeSize: The stripe size, in bytes, of the new volume.<69>

ulAlign: The number of bytes for the volume alignment. If zero is specified, the server determines the alignment value based on the size of the disk on which the volume is created.<70>

ppAsync: A pointer to an IVdsAsync interface that, if the operation is successfully completed, receives the IVdsAsync interface to monitor and control this operation. Callers MUST release the interface when they are done with it. If the **IVdsAsync::Wait (Opnum 4)** method is called on the interface, the interfaces returned in the **VDS_ASYNC_OUTPUT** structure MUST be released as well. For more information on handling asynchronous tasks, see section 3.3.5.1.7.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

IVdsPack2::CreateVolume2 has the same sequencing rules as **IVdsPack::CreateVolume (Opnum 7)**, as specified in section 3.3.5.2.16.5.

3.3.5.2.18 IVdsDisk Methods

3.3.5.2.18.1 IVdsDisk::GetProperties (Opnum 3)

The **GetProperties** method retrieves the properties of the disk that the object exposing this interface and method represents.

```
HRESULT GetProperties(
  [out] VDS_DISK_PROP* pDiskProperties
);
```

pDiskProperties: A pointer to a **VDS_DISK_PROP** structure that, if the operation is successfully completed, receives the properties of the disk.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **pDiskProperties** is not NULL.

The server MUST populate the **VDS_DISK_PROP** structure that **pDiskProperties** references with the properties of the disk; it MUST then return an HRESULT indicating failure or success. For information on VDS_DISK_PROP, see section 2.2.2.8.2.1.

3.3.5.2.18.2 IVdsDisk::GetPack (Opnum 4)

The **GetPack** method retrieves the disk pack that the disk belongs to.

```
HRESULT GetPack(
  [out] IVdsPack** ppPack
);
```

ppPack: A pointer to an **IVdsPack** interface that, if the operation is successfully completed, receives the **IVdsPack** interface of the pack object that the disk belongs to. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppPack** is not NULL.

The server MUST point **ppPack** to an **IVdsPack**interface of the pack object that the disk object's pack pointer refers to, and then return an HRESULT indicating failure or success.

3.3.5.2.18.3 IVdsDisk::GetIdentificationData (Opnum 5)

The **GetIdentificationData** method retrieves information that uniquely identifies a disk.

```
HRESULT GetIdentificationData(
  [out] VDS_LUN_INFORMATION* pLunInfo
);
```

pLunInfo: A pointer to a **VDS_LUN_INFORMATION** structure that, if the operation is successfully completed, receives the LUN information for the disk.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **pLunInfo** is not NULL.

The server MUST populate the **VDS_LUN_INFORMATION** structure that **pLunInfo** references with the LUN information that uniquely identifies the disk; it MUST then return an HRESULT indicating failure or success. For information on VDS_LUN_INFORMATION, see section 2.2.1.3.16.

3.3.5.2.18.4 IVdsDisk::QueryExtents (Opnum 6)

The **QueryExtents** method enumerates a disk's extents.

```
HRESULT QueryExtents(
  [out, size_is(,*plNumberOfExtents)]
    VDS_DISK_EXTENT** ppExtentArray,
  [out] long* plNumberOfExtents
);
```

ppExtentArray: A pointer to an array of **VDS_DISK_EXTENT** structures that, if the operation is successfully completed, receives the array of disk extent properties.

plNumberOfExtents: A pointer to a variable that, if the operation is successfully completed, receives the total number of elements in **ppExtentArray**.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **ppExtentArray** is not NULL.
- Verify that **plNumberOfExtents** is not NULL.

The server MUST point **ppExtentArray** to an array of **VDS_DISK_EXTENT** structures containing information about each disk extent on the disk, point **plNumberOfExtents** to the number of elements in the array, and return an HRESULT indicating failure or success.

For removable media drives, the server MUST set the value of **volumeId** and **plexId** in the VDS_DISK_EXTENT structure to the VDS_OBJECT_ID of the volume and volume plex object associated with the drive.

For removable media drive with no media, the server MUST return a single extent of type VDS_DET_UNKNOWN with the values of **ullOffset** and **ullSize** set to 0.

3.3.5.2.18.5 IVdsDisk::ConvertStyle (Opnum 7)

The **ConvertStyle** method converts a disk's partitioning format.

```
HRESULT ConvertStyle(
  [in] VDS_PARTITION_STYLE NewStyle
);
```

NewStyle: A value from the **VDS_PARTITION_STYLE** enumeration that indicates the new partitioning format.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST convert the disk's partitioning format to the style **NewStyle** specifies, and then return an HRESULT indicating failure or success.

3.3.5.2.18.6 IVdsDisk::SetFlags (Opnum 8)

The **SetFlags** method sets the read-only flag of a disk.

```
HRESULT SetFlags(
  [in] unsigned long ulFlags
);
```

ulFlags: MUST be set to VDS_DF_READ_ONLY.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that the **ulFlags** parameter is VDS_DF_READ_ONLY.

The server MUST perform the following in sequence:

- Set the read-only attribute of the disk.

- For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3))** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_DISK.
  - **Disk** member is a **VDS_DISK_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_DISK_MODIFY.
    - **diskId** is the VDS_OBJECT_ID of this disk object.
- Return an HRESULT indicating failure or success.

3.3.5.2.18.7 IVdsDisk::ClearFlags (Opnum 9)

The **ClearFlags** method clears the read only flag of a disk.

```
HRESULT ClearFlags(
  [in] unsigned long ulFlags
);
```

ulFlags: MUST be set to VDS_DF_READ_ONLY.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that the **ulFlags** parameter is VDS_DF_READ_ONLY.

The server MUST perform the following in sequence:

- Clear the read-only attribute of the disk.
- For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_DISK.
  - **Disk** member is a **VDS_DISK_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_DISK_MODIFY.
    - **diskId** is the VDS_OBJECT_ID of this disk object.
- Return an HRESULT indicating failure or success.

3.3.5.2.19 IVdsDisk2 Methods

3.3.5.2.19.1 IVdsDisk2::SetSANMode (Opnum 3)

The **SetSANMode** method sets the SAN mode of a disk to either offline (read-only mode) or online (read/write mode).<71>

```
HRESULT SetSANMode(
  [in] long bEnable
);
```

bEnable: A Boolean value that indicates whether to set the disk to either online or offline.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST set the SAN mode of the disk to offline or online, as specified by **bEnable**, and then return an HRESULT indicating failure or success.

3.3.5.2.20 IVdsAdvancedDisk Methods

3.3.5.2.20.1 IVdsAdvancedDisk::GetPartitionProperties (Opnum 3)

The **GetPartitionProperties** method retrieves the properties of a partition on the disk at a specified byte offset.

```
HRESULT GetPartitionProperties(
  [in] ULONGLONG ullOffset,
  [out] VDS_PARTITION_PROP* pPartitionProp
);
```

ullOffset: The byte offset of the partition, from the beginning of the disk. This offset MUST be the offset of a start of a partition.

pPartitionProp: A pointer to a **VDS_PARTITION_PROP** structure that, if the operation is successfully completed, receives the properties of the partition.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **pPartitionProp** is not NULL.

The server MUST populate the **VDS_PARTITION_PROP** structure that **pPartitionProp** references with the properties of the partition at the byte offset from the beginning of the disk that **ullOffset** specifies. The server MUST then return an HRESULT indicating failure or success. For information on VDS_PARTITION_PROP, see section 2.2.1.3.20.

3.3.5.2.20.2 IVdsAdvancedDisk::QueryPartitions (Opnum 4)

The **QueryPartitions** method enumerates a disk's partitions.

```
HRESULT QueryPartitions(
  [out, size_is(,*plNumberOfPartitions)]
    VDS_PARTITION_PROP** ppPartitionPropArray,
  [out] long* plNumberOfPartitions
```

```
);
```

ppPartitionPropArray: A pointer to an array of **VDS_PARTITION_PROP** structures that, if the operation is successfully completed, receives the array of partition properties.

plNumberOfPartitions: A pointer to a variable that, if the operation is successfully completed, receives the total number of elements in **ppPartitionPropArray**.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **ppPartitionPropArray** is not NULL.
- Verify that **plNumberOfPartitions** is not NULL.

The server MUST point **ppPartitionPropArray** to an array of **VDS_PARTITION_PROP** structures that contain information about each partition on the disk, point **plNumberOfPartitions** to the number of elements in the array, and then return an HRESULT indicating failure or success.

3.3.5.2.20.3 IVdsAdvancedDisk::CreatePartition (Opnum 5)

The **CreatePartition** method creates a partition on a disk at a specified byte offset.

```
HRESULT CreatePartition(
  [in] ULONGLONG ullOffset,
  [in] ULONGLONG ullSize,
  [in] CREATE_PARTITION_PARAMETERS* para,
  [out] IVdsAsync** ppAsync
);
```

ullOffset: MUST be the byte offset from the beginning of the disk at which to create the new partition.

ullSize: MUST be the size of the new partition, in bytes.

para: MUST be a pointer to a **CREATE_PARTITION_PARAMETERS** structure that describes the new partition to create.

ppAsync: MUST be a pointer to an **IVdsAsync** interface that, upon successful completion, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface received when they are done with it. If the **IVdsAsync::Wait** method is called on the interface, the interfaces returned in the VDS_ASYNC_OUTPUT structure MUST be released as well.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that para is not NULL.
- Verify that ppAsync is not NULL.

The server MUST perform the following:

- Create a new async object implementing the **IVdsAsync** interface with an output type of VDS_ASYNCOUT_CREATEPARTITION and set the pointer that **ppAsync** references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Create a new partition following the parameters specified to the method.
  - If the partition creation resulted in the renumbering of existing partitions on a boot disk, the server MUST update the boot configuration file with the new partition numbering.
- Set the task-specific return values in the async object to return the values associated with VDS_ASYNCOUT_CREATEPARTITION. For information on asynchronous tasks, see section 3.3.5.1.7.
- Set the return code in the async object to an HRESULT indicating failure or success.
  - If the server was required to update the boot configuration file but failed, the return code MUST be set to VDS_S_UPDATE_BOOTFILE_FAILED (HRESULT of 0x80042434).
- If the task was completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.
- For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - objectType member is VDS_NTT_PARTITION.
  - Partition member is a VDS_PARTITION_NOTIFICATION that has the following attributes:
    - ulEvent is VDS_NF_PARTITION_ARRIVE.
    - diskId is the VDS_OBJECT_ID of the disk object that corresponds to the disk on which the partition was added.
    - ullOffset is the byte offset at which the partition starts on the disk.
- If the partition is created on a removable media disk, for each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Volume** member is a **VDS_VOLUME_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the **VDS_OBJECT_ID** of the volume object corresponding to the removable media drive.

- For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)**) method by using a VDS_NOTIFICATION structure that has the following attributes:
  - **objectType** member is VDS_NTT_DISK.
  - **Disk** member is a **VDS_DISK_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_DISK_MODIFY.
    - **diskId** is the **VDS_OBJECT_ID** of this disk object.

At any point in the previous sequence—before the percentage completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.20.4 IVdsAdvancedDisk::DeletePartition (Opnum 6)

The **DeletePartition** method deletes a partition from the disk at a specified byte offset.

```
HRESULT DeletePartition(
  [in] ULONGLONG ullOffset,
  [in] long bForce,
  [in] long bForceProtected
);
```

ullOffset: The byte offset of the partition, from the beginning of the disk. This offset MUST be the offset at the start of a partition.

bForce: A Boolean that determines whether the partition deletion is forced.

bForceProtected: A Boolean value that determines whether deletion of a protected partition will be forced.<72>

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST perform the following:

- Delete the partition following the parameters specified to the method.
- If deleting the partition removed a volume, remove the corresponding volume object from the list of storage management objects. For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Volume** member is a VDS_VOLUME_NOTIFICATION that has the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_DEPART.
    - **volumeId** is the VDS_OBJECT_ID of the volume object that was removed.

- For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_PARTITION.
  - Partition member is a **VDS_PARTITION_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_PARTITION_DEPART.
    - **diskId** is the VDS_OBJECT_ID of the disk object corresponding to the disk from which the partition was deleted.
    - **ullOffset** is the byte offset at which the partition started on the disk.
- For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_DISK.
  - **Disk** member is a **VDS_DISK_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_DISK_MODIFY.
    - **diskId** is the VDS_OBJECT_ID of this disk object.
- Return an HRESULT indicating failure or success.

3.3.5.2.20.5 IVdsAdvancedDisk::ChangeAttributes (Opnum 7)

The **ChangeAttributes** method changes the attributes of the partition at byte offset *ullOffset* on the disk.

```
HRESULT ChangeAttributes(
  [in] ULONGLONG ullOffset,
  [in] CHANGE_ATTRIBUTES_PARAMETERS* para
);
```

ullOffset: The byte offset of the partition, from the beginning of the disk. This offset MUST be the offset of the start of a partition.

para: A pointer to a **CHANGE_ATTRIBUTES_PARAMETERS** structure that describes the attributes to change.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **para** is not NULL.
- Verify that the partition format in input parameter **para** matches the current partition format of the disk.

The server MUST perform the following:

- Change the attributes of the partition following the parameters specified to the method. If the disk partitioning format is MBR, the only value that may be changed is the bootIndicator. If the disk partitioning format is GPT, the only value that may be changed is the GPT attributes. The disk partitioning format may not be changed using this method. For information on changing partition attributes, see section 2.2.2.9.1.1.
- If attributes on the partition were successfully changed, for each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_PARTITION.
  - **Partition** member is a **VDS_PARTITION_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_PARTITION_MODIFY.
    - **diskId** is the VDS_OBJECT_ID of the disk object corresponding to the disk on which the partition attribute was modified.
    - **ullOffset** is the byte offset where the partition started on the disk.
- If attributes on the partition were successfully changed, and a volume exists on the partition, for each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Volume** member is a **VDS_VOLUME_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the VDS_OBJECT_ID of this volume object.
- Return an HRESULT indicating failure or success.

3.3.5.2.20.6 IVdsAdvancedDisk::AssignDriveLetter (Opnum 8)

The **AssignDriveLetter** method assigns a drive letter to an existing OEM, ESP, or unknown partition.

```
HRESULT AssignDriveLetter(
  [in] ULONGLONG ullOffset,
  [in] WCHAR wcLetter
);
```

ullOffset: The byte offset of the partition, from the beginning of the disk. This offset MUST be the offset of a start of a partition.

wcLetter: The drive letter to assign, as a single uppercase or lowercase alphabetical (A-Z) Unicode character.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that the partition at the byte offset specified by *ullOffset* does not have a volume existing on it.
- Verify that the drive letter specified by *wcLetter* is not already used.

The server MUST assign the drive letter to the partition and return an HRESULT indicating failure or success.

3.3.5.2.20.7 IVdsAdvancedDisk::DeleteDriveLetter (Opnum 9)

The **DeleteDriveLetter** method deletes a drive letter that is assigned to an OEM, ESP, or unknown partition.

```
HRESULT DeleteDriveLetter(
  [in] ULONGLONG ullOffset,
  [in] WCHAR wcLetter
);
```

ullOffset: The byte offset of the partition from the beginning of the disk. This offset MUST be the offset of a start of a partition.

wcLetter: The drive letter to delete as a single uppercase or lowercase alphabetical (A-Z) Unicode character.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that the partition at the byte offset that is specified by *ullOffset* does not have a volume existing on it.
- Verify that the partition is using the drive letter specified by *wcLetter*.

The server MUST delete the drive letter from the partition and return an HRESULT indicating failure or success.

3.3.5.2.20.8 IVdsAdvancedDisk::GetDriveLetter (Opnum 10)

The **GetDriveLetter** method retrieves the drive letter of a partition on the disk at a specified byte offset.

```
HRESULT GetDriveLetter(
  [in] ULONGLONG ullOffset,
  [out] WCHAR* pwcLetter
);
```

ullOffset: The byte offset of the partition, from the beginning of the disk. This offset MUST be the offset of a start of a partition.

pwcLetter: A pointer to a Unicode character that will receive an uppercase or lowercase alphabetical (A-Z) drive letter for the partition at byte offset *ullOffset*.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST validate the following parameters:

- Verify that the partition at the byte offset that is specified by *ullOffset* does not have a volume existing on it.
- Verify that the partition has a drive letter.
- Verify that *pwcLetter* is not NULL.

The server MUST set a value referenced by *pwcLetter* with the drive letter of the partition and return an HRESULT indicating failure or success.

3.3.5.2.20.9 IVdsAdvancedDisk::FormatPartition (Opnum 11)

The **FormatPartition** method formats an existing OEM, ESP, or unknown partition.

```
HRESULT FormatPartition(
  [in] ULONGLONG ullOffset,
  [in] VDS_FILE_SYSTEM_TYPE type,
  [in, string] WCHAR* pwszLabel,
  [in] DWORD dwUnitAllocationSize,
  [in] long bForce,
  [in] long bQuickFormat,
  [in] long bEnableCompression,
  [out] IVdsAsync** ppAsync
);
```

ullOffset: The byte offset of the partition, from the beginning of the disk. This offset MUST be the offset of a start of a partition.

type: A file system type that is enumerated by **VDS_FILE_SYSTEM_TYPE**. Clients that want to format by using file systems that are not enumerated by **VDS_FILE_SYSTEM_TYPE**, may use the **IVdsDiskPartitionMF::FormatPartionEx** method.

pwszLabel: A null-terminated Unicode string representing the partition label. The maximum label size is file system-dependent.

dwUnitAllocationSize: The size, in bytes, of the allocation unit for the file system. The value MUST be a power of 2. Allocation unit range is file system-dependent.

bForce: A Boolean that determines whether the format is forced, regardless of whether the volume is in use.

bQuickFormat: A Boolean that determines whether a file system is **quick formatted**. A quick format does not verify each sector on the volume.

bEnableCompression: A Boolean that determines whether a file system is created with compression enabled. <73>

ppAsync: A pointer to an IVdsAsync interface that, if the operation is successfully completed, receives the IVdsAsync interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppAsync* is not NULL.

The server MUST perform the following:

- Create a new async object implementing the IVdsAsync interface by using an output type of VDS_ASYNCOUT_FORMAT and set the pointer that *ppAsync* references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Format the partition following the parameters specified to the method.
- If TRUE is specified for **bEnableCompression** and the file system being formatted is an NTFS file system, compress the file system after formatting is complete.
- Set the return code in the async object to an HRESULT indicating failure or success.
  - If the server tried to compress the file system after formatting it and failed, the return code MUST be set to VDS_S_VOLUME_COMPRESS_FAILED— an HRESULT of 0x00042443.
- If the task completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage completed value in the async object is 100—the server MAY update the percentage completed value.

3.3.5.2.20.10 IVdsAdvancedDisk::Clean (Opnum 12)

The **Clean** method cleans a disk.

```
HRESULT Clean(
  [in] long bForce,
  [in] long bForceOEM,
  [in] long bFullClean,
  [out] IVdsAsync** ppAsync
);
```

bForce: A Boolean value that indicates whether the cleaning operation will be forced. If set, the method attempts to clean the disk, even if data volumes or ESP partitions are present.

bForceOEM: A Boolean value that indicates whether the cleaning operation of an OEM partition will be forced. If the disk contains an OEM partition but **bForceOEM** is not set, the operation

SHOULD fail. If the value is set, the method attempts to clean the disk, even if OEM partitions are present.<74>

bFullClean: A Boolean value that indicates whether the cleaning operation removes all the data from the disk.

ppAsync: A pointer to an IVdsAsync interface that, if the operation is successfully completed, receives the IVdsAsync interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppAsync** is not NULL.

The server MUST perform the following:

- Create a new async object implementing the IVdsAsync interface with an output type of VDS_ASYNCOUT_CLEAN and set the pointer referenced by *ppAsync* to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Clean the disk, removing all partition information, following the parameters specified to the method.
- Set the return code in the async object to an HRESULT indicating failure or success.
  - If TRUE was specified for *bFullClean* but the server could not clean particular sectors, the return code MUST be set to VDS_S_DISK_PARTIALLY_CLEANED.
- If the task was completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.
- If the disk is a removable media disk, for each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method with a VDS_NOTIFICATION structure with the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Volume** member is a **VDS_VOLUME_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the VDS_OBJECT_ID of the volume object corresponding to the removable media drive.
- For each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - **objectType** member is VDS_NTT_DISK.
  - **Disk** member is a **VDS_DISK_NOTIFICATION** with the following attributes:

- **ulEvent** is VDS_NF_DISK_MODIFY.
- **diskId** is the **VDS_OBJECT_ID** of this disk object.

At any point in the previous sequence—before the percentage completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.21 IVdsAdvancedDisk2 Methods

3.3.5.2.21.1 IVdsAdvancedDisk2::ChangePartitionType (Opnum 3)

The **ChangePartitionType** method changes the partition type on the disk at a specified byte offset.

```
HRESULT ChangePartitionType(
  [in] ULONGLONG ullOffset,
  [in] long bForce,
  [in] CHANGE_PARTITION_TYPE_PARAMETERS* para
);
```

ullOffset: The byte offset of the partition, from the beginning of the disk. This offset MUST be the offset of a start of a partition.

bForce: A Boolean value that indicates whether change will be forced.

para: A pointer to a **CHANGE_PARTITION_TYPE_PARAMETERS** structure that contains the partition type that the partition at the location specified by *ullOffset* is changed to.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST validate the following parameter:

- Verify that para is not NULL.

The server MUST perform the following:

- Change the partition type following the parameters specified to the method.
- If a volume exists on the partition, for each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify** method with a VDS_NOTIFICATION structure with the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Volume** member is a VDS_VOLUME_NOTIFICATION with the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the VDS_OBJECT_ID of this volume object.
- Return an HRESULT indicating failure or success.

3.3.5.2.22 IVdsCreatePartitionEx Methods

3.3.5.2.22.1 IVdsCreatePartitionEx::CreatePartitionEx (Opnum 3)

The **CreatePartitionEx** method creates a partition on a disk at a specified byte offset, with an optional alignment parameter.

```
HRESULT CreatePartitionEx(
  [in] ULONGLONG ullOffset,
  [in] ULONGLONG ullSize,
  [in] unsigned long ulAlign,
  [in] CREATE_PARTITION_PARAMETERS* para,
  [out] IVdsAsync** ppAsync
);
```

ullOffset: The byte offset from the beginning of the disk where the new partition will be created. If *ulAlign* is not zero, the offset MUST fall within the first cylinder for an MBR disk (GPT disks do not have this restriction).

ullSize: The size of the new partition, in bytes.<75>

ppAsync: A pointer to an IVdsAsync interface that, if the operation is successfully completed, receives the IVdsAsync interface to monitor and control this operation. Callers MUST release the interface when they are done with it. If the **IVdsAsync::Wait** method is called on the interface, the interfaces returned in the **VDS_ASYNC_OUTPUT** structure MUST be released as well. For information on asynchronous tasks, see section 3.3.5.1.7.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST validate the following parameters:

- Verify that *para* is not NULL.
- Verify that *ppAsync* is not NULL.

The server MUST perform the following:

- Create a new async object implementing the IVdsAsync interface with an output type of VDS_ASYNCOUT_CREATEPARTITION and set the pointer *ppAsync* references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Create a new partition following the parameters specified to the method.
  - If the partition creation resulted in the renumbering of existing partitions on a boot disk, the server MUST update the boot configuration file with the new partition numbering.
- Set the task-specific return values in the async object to return the values associated with VDS_ASYNCOUT_CREATEPARTITION (as specified in section 3.3.5.1.7).

- Set the return code in the async object to an HRESULT indicating failure or success.
  - If the server was required to update the boot configuration file but failed, the return code MUST be set to VDS_S_UPDATE_BOOTFILE_FAILED (HRESULT of 0x80042434).
- If the task completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.
- For each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify** method with a VDS_NOTIFICATION structure with the following attributes:
  - objectType member is VDS_NTT_PARTITION.
  - Partition member is a VDS_PARTITION_NOTIFICATION with the following attributes:
    - ulEvent is VDS_NF_PARTITION_ARRIVE.
    - diskId is the VDS_OBJECT_ID of the disk object corresponding to the disk on which the partition was added.
    - ullOffset is the byte offset at which the partition starts on the disk.
- If the partition is created on a removable media disk, for each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify** method with a VDS_NOTIFICATION structure with the following attributes:
  - objectType member is VDS_NTT_VOLUME.
  - Volume member is a VDS_VOLUME_NOTIFICATION with the following attributes:
    - ulEvent is VDS_NF_VOLUME_MODIFY.
    - volumeId is the VDS_OBJECT_ID of the volume object corresponding to the removable media drive.
- For each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - objectType member is VDS_NTT_DISK.
  - **Disk** member is a **VDS_DISK_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_DISK_MODIFY.
    - **diskId** is the VDS_OBJECT_ID of this disk object.

At any point in the previous sequence—before the percentage completed value in the async object is 100—the server MAY update the percentage completed value.

3.3.5.2.23 IVdsDiskPartitionMF Methods

3.3.5.2.23.1 IVdsDiskPartitionMF::GetPartitionFileSystemProperties (Opnum 3)

The **GetPartitionFileSystemProperties** method returns property details about the file system on a disk partition at a specified byte offset.<76>

```
HRESULT GetPartitionFileSystemProperties(
  [in] ULONGLONG ullOffset,
  [out] VDS_FILE_SYSTEM_PROP* pFileSystemProp
);
```

ullOffset: The byte offset of the partition from the beginning of the disk. This MUST be the offset at the start of a partition.

pFileSystemProp: A pointer to a **VDS_FILE_SYSTEM_PROP** structure that, if the operation is successfully completed, receives the properties of the file system on the partition.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST validate the following parameter:

- Verify that *pFileSystemProp* is not NULL.

The server MUST populate the **VDS_FILE_SYSTEM_PROP** structure that *pFileSystemProp* references with the properties of the file system on the partition at the byte offset from the beginning of the disk that *ullOffset* specifies. The server MUST then return an HRESULT indicating failure or success. For more information on the VDS_FILE_SYSTEM_PROP structure, see section 2.2.1.3.17.

3.3.5.2.23.2 IVdsDiskPartitionMF::GetPartitionFileSystemTypeName (Opnum 4)

The **GetPartitionFileSystemTypeName** method retrieves the name of the file system on a disk partition at a specified byte offset.<77>

```
HRESULT GetPartitionFileSystemTypeName(
  [in] ULONGLONG ullOffset,
  [out, string] WCHAR** ppwszFileSystemTypeName
);
```

ullOffset: The byte offset of the partition from the beginning of the disk. This MUST be the offset at the start of a partition.

ppwszFileSystemTypeName: A pointer that, if the operation is successfully completed, receives a null-terminated Unicode string with the file system name.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppwszFileSystemTypeName** is not NULL.

The server MUST point **ppwszFileSystemTypeName** to a string containing the file system name on the partition at the byte offset from the beginning of the disk that *ullOffset* specifies, and then return an HRESULT indicating failure or success.

3.3.5.2.23.3 IVdsDiskPartitionMF::QueryPartitionFileSystemFormatSupport (Opnum 5)

The **QueryPartitionFileSystemFormatSupport** method retrieves the properties of the file systems that support formatting a disk partition at a specified byte offset.<78>

```
HRESULT QueryPartitionFileSystemFormatSupport(
  [in] ULONGLONG ullOffset,
  [out, size_is(,*plNumberOfFileSystems)]
    VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP** ppFileSystemSupportProps,
  [out] long* plNumberOfFileSystems
);
```

ullOffset: The byte offset of the partition from the beginning of the disk. This MUST be the offset at the start of a partition.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST validate the following parameters:

- Verify that **ppFileSystemSupportProps** is not NULL.
- Verify that **plNumberOfFileSystems** is not NULL.

The server MUST point **ppFileSystemSupportProps** to an array of **VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP** structures containing information about each file system that supports formatting a partition at the byte offset from the beginning of the disk that *ullOffset* specifies. The server MUST then point **plNumberOfFileSystems** to the size of the array, and then return an HRESULT indicating failure or success.

3.3.5.2.23.4 IVdsDiskPartitionMF::FormatPartitionEx (Opnum 6)

The **FormatPartitionEx** method formats an existing OEM, ESP, or unknown partition.<79>

```
HRESULT FormatPartitionEx(
  [in] ULONGLONG ullOffset,
  [in, unique, string] WCHAR* pwszFileSystemTypeName,
  [in] unsigned short usFileSystemRevision,
  [in] unsigned long ulDesiredUnitAllocationSize,
  [in, unique, string] WCHAR* pwszLabel,
  [in] long bForce,
  [in] long bQuickFormat,
  [in] long bEnableCompression,
  [out] IVdsAsync** ppAsync
```

```
);
```

ullOffset: The byte offset of the partition from the beginning of the disk. This MUST be the offset at the start of a partition.

pwszFileSystemTypeName: A null-terminated Unicode string that contains the name of the file system with which to format the partition.

usFileSystemRevision: A 16-bit, binary-coded decimal number that indicates the revision of the file system, if any. The first two (most significant) digits (8-bits) indicate the major revision while the last two (least significant) digits (8-bits) indicate the minor revision (for example, 0x0250 represents revision 2.50).

ulDesiredUnitAllocationSize: The size, in bytes, of the allocation unit for the file system. The value MUST be a power of 2. If the value is 0, a default allocation unit determined by the file system type is used. The allocation unit range is file system-dependent.

pwszLabel: The null-terminated Unicode string to assign to the new file system. The maximum label size is file system-dependent.

bForce: A Boolean that determines whether a file system format is forced, even if the partition is in use.

bQuickFormat: A Boolean that determines whether a file system is quick formatted. A quick formatted does not verify each sector on the volume.

bEnableCompression: A Boolean that determines whether a file system is created with compression enabled.<80>

ppAsync: A pointer to an IVdsAsync interface that, if the operation is successfully completed, receives the IVdsAsync interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppAsync** is not NULL.

The server MUST perform the following:

- Create a new async object implementing the IVdsAsync interface with an output type of VDS_ASYNCOUT_FORMAT and set the pointer **ppAsync** references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Format the partition following the parameters specified to the method.
- If TRUE is specified for *bEnableCompression* and the file system being formatted is an NTFS file system, compress the file system after formatting is complete.

- Set the return code in the async object to an HRESULT indicating failure or success.
  - If the server was required to compress the file system after formatting but could not, the return code MUST be set to VDS_S_VOLUME_COMPRESS_FAILED (HRESULT of 0x00042443).
- If the task completed successfully, set the percentage completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage-completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.24 IVdsRemovable Methods

3.3.5.2.24.1 IVdsRemovable::QueryMedia (Opnum 3)

The **QueryMedia** method identifies the media in the drive. This method has no parameters.

```
HRESULT QueryMedia();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST perform the following:

- Refresh the cached disk properties of the removable media drive.
- If the cached disk properties for the media have changed, for each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - **objectType** member is VDS_NTT_DISK.
  - **Disk** member is a **VDS_DISK_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_DISK_MODIFY.
    - **diskId** is the **VDS_OBJECT_ID** of this disk object.
- If the cached disk properties for the media have changed, for each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Volume** member is a VDS_VOLUME_NOTIFICATION with the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.

- **volumeId** is the **VDS_OBJECT_ID** of the volume object corresponding to the removable media drive.

- Return an HRESULT indicating failure or success.

3.3.5.2.24.2 IVdsRemovable::Eject (Opnum 4)

The **Eject** method ejects the media in the drive. This method has no parameters.

```
HRESULT Eject();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST perform the following:

- Attempt to eject the media in the removable media drive.
- For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify (Opnum 3)** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Volume** member is a **VDS_VOLUME_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the **VDS_OBJECT_ID** of the volume object corresponding to the removable media drive.
- Return an HRESULT indicating failure or success.

3.3.5.2.25 IVdsVolume Methods

3.3.5.2.25.1 IVdsVolume::GetProperties (Opnum 3)

The **GetProperties** method retrieves the properties of the volume that is represented by the object exposing this interface and method.

```
HRESULT GetProperties(
  [out] VDS_VOLUME_PROP* pVolumeProperties
);
```

pVolumeProperties: A pointer to a **VDS_VOLUME_PROP** structure that, if the operation is successfully completed, receives the properties of the volume.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **pVolumeProperties** is not NULL.

The server MUST populate the **VDS_VOLUME_PROP** structure that **pVolumeProperties** references with the properties of the volume, and then return an HRESULT indicating failure or success. For information on VDS_VOLUME_PROP, see section 2.2.2.11.2.1.

3.3.5.2.25.2 IVdsVolume::GetPack (Opnum 4)

The **GetPack** method retrieves the disk pack to which the volume belongs.

```
HRESULT GetPack(
  [out] IVdsPack** ppPack
);
```

ppPack: A pointer to an *IVdsPack* interface that, if the operation is successfully completed, receives the *IVdsPack* interface of the pack object to which the volume belongs. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppPack** is not NULL.

The server MUST point **ppPack** to an *IVdsPack* interface of the pack object that the volume object's pack pointer refers to. It MUST then return an HRESULT indicating failure or success.

3.3.5.2.25.3 IVdsVolume::QueryPlexes (Opnum 5)

The **QueryPlexes** method enumerates the plexes of a volume.

```
HRESULT QueryPlexes(
  [out] IEnumVdsObject** ppEnum
);
```

ppEnum: A pointer to an **IEnumVdsObject** interface that, if the operation is successfully completed, receives the **IEnumVdsObject** interface of the object that contains an enumeration of volume plex objects in the volume. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppEnum** is not NULL.

The server MUST point **ppEnum** to an **IEnumVdsObject** interface that contains the enumeration of volume plex objects in the list of cached storage management objects that have their volume pointer set to this volume object, as specified in section 3.3.1.3. The server MUST then return an HRESULT indicating failure or success.

3.3.5.2.25.4 IVdsVolume::Extend (Opnum 6)

The **Extend** method expands the size of the current volume by adding disk extents to each member of each plex.

```
HRESULT Extend(
  [in, unique, size_is(lNumberOfDisks)]
    VDS_INPUT_DISK* pInputDiskArray,
  [in] long lNumberOfDisks,
  [out] IVdsAsync** ppAsync
);
```

pInputDiskArray: A pointer to an array of **VDS_INPUT_DISK** structures that describe the disk extents to add to the volume—one structure for each disk. Callers SHOULD specify the member index for all the disk extents together with the **Extend** method, unless the volume has only one plex with only one member.

lNumberOfDisks: The number of elements in *pInputDiskArrray*.<81>

ppAsync: A pointer to an **IVdsAsync** interface that, if the operation is successfully completed, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that *pInputDiskArray* is not NULL.
- Verify that *ppAsync* is not NULL.

The server MUST perform the following:

- Create a new async object implementing the **IVdsAsync**interface with an output type of VDS_ASYNCOUT_EXTENDVOLUME and set the pointer *ppAsync* references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Extend the volume following the parameters specified to the method.
- For each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify()** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - objectType member is VDS_NTT_VOLUME.
  - Volume member is a VDS_VOLUME_NOTIFICATION with the following attributes:
    - ulEvent is VDS_NF_VOLUME_MODIFY.
    - volumeId is the VDS_OBJECT_ID of this volume object.
- If the file system on the volume is an NTFS file system, extend the file system to fill the newly extended volume. For each callback object registered in the list of callback objects, call the

callback object's **IVdsAdviseSink::OnNotify()** method with a **VDS_NOTIFICATION** structure with the following attributes:

- **objectType** member is VDS_NTT_FILE_SYSTEM.
- **Volume** member is a **VDS_FILE_SYSTEM_NOTIFICATION** with the following attributes:
  - **ulEvent** is VDS_NF_FILE_SYSTEM_MODIFY.
  - **volumeId** is the VDS_OBJECT_ID of this volume object.

- Set the return code in the async object to an HRESULT indicating failure or success.
- If the task completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage-completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.25.5 IVdsVolume::Shrink (Opnum 7)

The **Shrink** method reduces the size of the volume and all plexes, and returns the released extents to free space.<82>

```
HRESULT Shrink(
  [in] ULONGLONG ullNumberOfBytesToRemove,
  [out] IVdsAsync** ppAsync
);
```

ppAsync: A pointer to an **IVdsAsync** interface that, if the operation is successfully completed, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that *ppAsync* is not NULL.<83>

The server MUST perform the following:

- Create a new async object implementing the **IVdsAsync** interface with an output type of VDS_ASYNCOUT_SHRINKVOLUME and set the pointer *ppAsync* references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- If the file system on the volume is an NTFS file system, shrink the file system following the parameters specified to the method.
- Shrink the volume following the parameters specified to the method.

- For each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify()** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - objectType member is VDS_NTT_VOLUME.
  - The volume member is a **VDS_VOLUME_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the **VDS_OBJECT_ID** of this volume object.
- Set the return code in the async object to an HRESULT indicating failure or success.
- If the task completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage-completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.25.6 IVdsVolume::AddPlex (Opnum 8)

The **AddPlex** method adds a volume as a plex to the current volume.

```
HRESULT AddPlex(
  [in] VDS_OBJECT_ID VolumeId,
  [out] IVdsAsync** ppAsync
);
```

VolumeId: The VDS object ID of the volume object to add as a plex.

ppAsync: A pointer to an **IVdsAsync** interface that, if the operation is successfully completed, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that the volume corresponding to the **VDS_OBJECT_ID** that VolumeId specifies has only one volume plex.
- Verify that **ppAsync** is not NULL.

If the volume resides on a basic disk, the server MUST return VDS_E_NOT_SUPPORTED (HRESULT of 0x80042400).

The server MUST perform the following:

- Create a new async object implementing the **IVdsAsync** interface with an output type of VDS_ASYNCOUT_ADDVOLUMEPLEX and set the pointer that *ppAsync* references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Add the volume plex of the volume corresponding to the **VDS_OBJECT_ID** that VolumeId specifies as a volume plex to this volume.
- Set the volume plex's volume pointer to this volume object.
- Remove the volume object corresponding to the **VDS_OBJECT_ID** that VolumeId specifies from the list of storage management objects.
- Set the return code in the async object to an HRESULT indicating failure or success.
- If the task completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage-completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.25.7 IVdsVolume::BreakPlex (Opnum 9)

The **BreakPlex** method removes a specified plex from the current volume. The interface pointer for the new volume object can be retrieved by calling **IVdsAsync::Wait** through the *ppAsync* parameter. The **VDS_ASYNC_OUTPUT** structure that is returned contains the volume object interface pointer in the bvp.pVolumeUnk member.

```
HRESULT BreakPlex(
  [in] VDS_OBJECT_ID plexId,
  [out] IVdsAsync** ppAsync
);
```

plexId: The GUID of the plex to be broken.

ppAsync: A pointer to an **IVdsAsync** interface that, if the operation is successfully completed, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface when they are done with it. If the **IVdsAsync::Wait** method is called on the interface, the interfaces returned in the **VDS_ASYNC_OUTPUT** structure MUST be released as well. For information on asynchronous tasks, see section 3.3.5.1.7.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppAsync** is not NULL.

The server MUST perform the following:

- Create a new async object implementing the **IVdsAsync** interface with an output type of VDS_ASYNCOUT_BREAKVOLUMEPLEX and set the pointer that **ppAsync** references to the interface.
- Return an HRESULT indicating failure or success.

If the volume resides on a basic disk, the server MUST return VDS_E_NOT_SUPPORTED (HRESULT of 0x80042400).

The server MUST then perform the following in sequence:

- Break the volume plex corresponding to the **VDS_OBJECT_ID** that **PlexId** specifies for this volume.
- Create a new volume object that corresponds to the new volume, implements the **IVdsVolume** interface, and assigns it a unique **VDS_OBJECT_ID**.
- Set the new volume object's pack pointer to the pack object that this volume's pack pointer references.
- Add the new volume object to the list of storage management objects.
- Set the volume plex's volume pointer to the new volume object. Set the task-specific return values in the async object to return the values that are associated with VDS_ASYNCOUT_BREAKVOLUMEPLEX (as specified in section 3.3.5.1.7).
- Set the return code in the async object to an HRESULT indicating failure or success.
- If the task completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage-completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.25.8 IVdsVolume::RemovePlex (Opnum 10)

The **RemovePlex** method removes a specified plex from a volume. The last plex of a volume cannot be removed.

```
HRESULT RemovePlex(
  [in] VDS_OBJECT_ID plexId,
  [out] IVdsAsync** ppAsync
);
```

plexId: The VDS object ID of the volume plex object to remove.

ppAsync: A pointer to an **IVdsAsync** interface that, if the operation is successfully completed, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppAsync** is not NULL.

The server MUST perform the following:

- Create a new async object implementing the **IVdsAsync**interface with an output type of VDS_ASYNCOUT_REMOVEVOLUMEPLEX and set the pointer that **ppAsync** references to the interface.
- Return an HRESULT indicating failure or success.

If the volume resides on a basic disk, the server MUST return VDS_E_NOT_SUPPORTED (HRESULT of 0x80042400).

The server MUST then perform the following in sequence:

- Remove the volume plex that corresponds to the **VDS_OBJECT_ID** **PlexId** specifies from this volume.
- Remove the corresponding volume plex object from the list of storage management objects.
- Set the return code in the async object to an HRESULT indicating failure or success.
- If the task completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage-completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.25.9 IVdsVolume::Delete (Opnum 11)

The **Delete** method deletes all plexes in a volume.<84>

```
HRESULT Delete(
  [in] long bForce
);
```

bForce: A Boolean that determines whether all plexes in a volume are deleted when the volume is in use.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST perform the following:

- Delete all volume plexes in the volume. Remove the corresponding volume plex objects from the list of storage management objects.
- Remove this volume object from the list of storage management objects.
- For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify()** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - objectType member is VDS_NTT_VOLUME.
  - Volume member is a **VDS_VOLUME_NOTIFICATION** with the following attributes:

- ulEvent is VDS_NF_VOLUME_DEPART.
- volumeId is the **VDS_OBJECT_ID** of this volume.

- If the volume resided on a basic disk disk, for each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify()** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - objectType member is VDS_NTT_PARTITION.
  - Partition member is a VDS_PARTITION_NOTIFICATION with the following attributes:
    - ulEvent is VDS_NF_PARTITION_DEPART.
    - diskId is the **VDS_OBJECT_ID** of the disk object on which the volume resided.
    - ullOffset is the byte offset at which the volume's partition started on the disk.
- If the volume resides on a basic disk and if the partition the volume resides on is the last remaining partition in an extended partition, delete the extended partition as well. Then for each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify()** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - objectType member is VDS_NTT_PARTITION.
  - Partition member is a **VDS_PARTITION_NOTIFICATION** with the following attributes:
    - ulEvent is VDS_NF_PARTITION_DEPART.
    - diskId is the **VDS_OBJECT_ID** of the disk object on which the extended partition resided.
    - ullOffset is the byte offset at which the extended partition started on the disk.
- If the volume resided on a basic disk, for each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify()** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - **objectType** member is VDS_NTT_DISK.
  - **Disk** member is a **VDS_DISK_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_DISK_MODIFY.
    - **diskId** is the **VDS_OBJECT_ID** of the disk object on which the volume resided.
- Return an HRESULT indicating failure or success.

3.3.5.2.25.10 IVdsVolume::SetFlags (Opnum 12)

The **SetFlags** method assigns flags to a volume.

```
HRESULT SetFlags(
  [in] unsigned long ulFlags,
  [in] long bRevertOnClose
);
```

ulFlags: The combination of any values, by using a bitwise OR operator, that are defined in the **VDS_VOLUME_FLAG** enumeration.

bRevertOnClose: A Boolean that determines whether the flags should be temporarily set. If they are temporarily set, VDS_VF_READONLY, VDS_VF_HIDDEN, VDS_VF_NO_DEFAULT_DRIVE_LETTER, and VDS_VF_SHADOW_COPY are the only valid flags that can be set, and the server reverts the flags after the client releases its last reference to the volume object.<85><86>

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **ulFlags** contains only those valid flags defined in the **VDS_VOLUME_FLAG** enumeration. If **bRevertOnClose** is set, then verify that **ulFlags** only contains VDS_VF_READONLY, VDS_VF_HIDDEN, VDS_VF_NO_DEFAULT_DRIVE_LETTER, and/or VDS_VF_SHADOW_COPY.
- If **bRevertOnClose** is set, verify that the volume object does not have flags that were previously set with **bRevertOnClose** and are yet to be reverted on close.
- If **bRevertOnClose** is not set, verify that the flags being set do not contain a flag that was previously set with **bRevertOnClose** and is yet to be reverted on close.

The server MUST perform the following:

- Set the volume flags specified by **ulFlags**.
- If **bRevertOnClose** is set, the server MUST be prepared to automatically revert the volume flags if a client releases the last reference to the volume object.
- For each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify()** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Volume** member is a **VDS_VOLUME_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the **VDS_OBJECT_ID** of this volume object.
- Return an HRESULT indicating failure or success.

3.3.5.2.25.11 IVdsVolume::ClearFlags (Opnum 13)

The **ClearFlags** method clears flags from a volume.

```
HRESULT ClearFlags(
  [in] unsigned long ulFlags
);
```

ulFlags: The combination of any values, by using the bitwise OR operator, that are defined in the **VDS_VOLUME_FLAG** enumeration.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **ulFlags** contains only valid flags defined in the **VDS_VOLUME_FLAG** enumeration.
- If volume flags were set previously by calling **SetFlags** with **bRevertOnClose** set, and those flags have not yet been reverted, verify that the flags that **ulFlags** specifies are exactly the same as the flags set by a call to **SetFlags**.

The server MUST perform the following:

- Clear the volume flags that **ulFlags** specifies. If the flags being cleared were set temporarily by calling **SetFlags** with **bRevertOnClose** set, and those flags had not yet been reverted, the server SHOULD NOT revert the flags automatically when a client releases the last reference to the volume object or dismounts the volume.
- For each callback object that is registered in the list of callback objects, call the **IVdsAdviseSink::OnNotify()** method of the callback object by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Volume** member is a **VDS_VOLUME_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the **VDS_OBJECT_ID** of this volume object.
- Return an HRESULT indicating failure or success.

3.3.5.2.26 IVdsVolumeMF Methods

3.3.5.2.26.1 IVdsVolumeMF::GetFileSystemProperties (Opnum 3)

The **GetFileSystemProperties** method returns property details about the file system on the current volume.

```
HRESULT GetFileSystemProperties(
  [out] VDS_FILE_SYSTEM_PROP* pFileSystemProp
);
```

pFileSystemProp: A pointer to a **VDS_FILE_SYSTEM_PROP** structure that, if the operation is successfully completed, receives the properties of the file system on the volume.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **pFileSystemProp** is not NULL.

The server MUST populate the **VDS_FILE_SYSTEM_PROP** structure (as specified in section 2.2.1.3.17) that **pFileSystemProp** references by using the properties of the file system on the volume. The server MUST then return an HRESULT indicating failure or success.

3.3.5.2.26.2 IVdsVolumeMF::Format (Opnum 4)

The **Format** method formats a file system on the current volume.

```
HRESULT Format(
  [in] VDS_FILE_SYSTEM_TYPE type,
  [in, string] WCHAR* pwszLabel,
  [in] DWORD dwUnitAllocationSize,
  [in] long bForce,
  [in] long bQuickFormat,
  [in] long bEnableCompression,
  [out] IVdsAsync** ppAsync
);
```

type: A file system type that is enumerated by **VDS_FILE_SYSTEM_TYPE**. Clients that format by using file systems that are not enumerated by **VDS_FILE_SYSTEM_TYPE** may use the **IVdsVolumeMF2::FormatEx** method.

pwszLabel: A null-terminated Unicode label to assign to the new file system. The maximum label size is file system-dependent.

dwUnitAllocationSize: The size, in bytes, of the allocation unit for the file system. The value MUST be a power of 2. The allocation unit range is file system-dependent.

bForce: A Boolean that determines whether the format is forced, even if the volume is in use.

bQuickFormat: A Boolean that determines a file system is quick format. A quick format does not verify each sector on the volume.

bEnableCompression: A Boolean that determines whether a file system is created with compression enabled.<87>

ppAsync: A pointer to an **IVdsAsync** interface that, if the operation is successfully completed, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppAsync** is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning a vendor-specific error as its response to the client.

The server MUST perform the following:

- Create a new async object implementing the **IVdsAsync** interface with an output type of VDS_ASYNCOUT_FORMAT and set the pointer referenced by **ppAsync** to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Format the volume following the parameters specified to the method.

- For each callback object registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify()** method with a **VDS_NOTIFICATION** structure with the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - Volume **member** is a **VDS_VOLUME_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the **VDS_OBJECT_ID** of this volume object.
- If TRUE is specified for **bEnableCompression**, and an NTFS file system is being formatted, compress the file system after formatting is complete.
- Set the return code in the async object to an HRESULT indicating failure or success.
  - If the server tried to compress the file system after formatting but failed, the return code MUST be set to VDS_S_VOLUME_COMPRESS_FAILED (HRESULT of 0x00042443).
- If the task completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.26.3 IVdsVolumeMF::AddAccessPath (Opnum 5)

The **AddAccessPath** method adds an access path to the current volume.<88>

```
HRESULT AddAccessPath(
  [in, max_is(MAX_PATH - 1), string]
    WCHAR* pwszPath
);
```

pwszPath: A null-terminated Unicode string that indicates the access path. If the access path is a drive letter, you must include a trailing backslash, for example, "F:\".

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **pwszPath** is not NULL.

The server MUST then perform the following in sequence:

- Add the access path to the volume.
- If the server determines that a mount point was added to the volume, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_MOUNT_POINT.

- **MountPoint** member is a **VDS_MOUNT_POINT_NOTIFICATION** with the following attributes:
  - **ulEvent** is VDS_NF_MOUNT_POINT_CHANGE.
  - **volumeId** is the **VDS_OBJECT_ID** of the volume object whose mount point was assigned.
- If the server determines that a drive letter was added to the volume, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_DRIVE_LETTER.
  - **Letter** member is a **VDS_DRIVE_LETTER_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_DRIVE_LETTER_ASSIGN.
    - **wcLetter** is the drive letter that was assigned to the volume.
    - **volumeId** is the **VDS_OBJECT_ID** of the volume object whose drive letter was assigned.
- Return an HRESULT indicating failure or success.

3.3.5.2.26.4 IVdsVolumeMF::QueryAccessPaths (Opnum 6)

The **QueryAccessPaths** method returns a list of access paths and a drive letter as a single case-insensitive Unicode character, if one exists, for the current volume.

```
HRESULT QueryAccessPaths(
  [out, string, size_is(,*plNumberOfAccessPaths)]
    WCHAR*** pppwszPathArray,
  [out] long* plNumberOfAccessPaths
);
```

pppwszPathArray: A pointer to an array of strings that, if the operation is successfully completed, receives the array of access paths.

plNumberOfAccessPaths: A pointer to a variable that, if the operation is successfully completed, receives the total number of elements returned in **ppwszPathArray**.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **ppwszPathArray** is not NULL.
- Verify that **plNumberOfAccessPaths** is not NULL.

If parameter validation fails, the server MUST fail the operation immediately, returning a vendor-specific error as its response to the client.

The server MUST point **ppwszPathArray** to an array of strings that contain the access paths to the volume, point **plNumberOfAccessPaths** to the size of the array, and return an HRESULT indicating failure or success.

3.3.5.2.26.5 IVdsVolumeMF::QueryReparsePoints (Opnum 7)

The **QueryReparsePoints** method returns all reparse points for the current volume.

```
HRESULT QueryReparsePoints(
  [out, size_is(,*plNumberOfReparsePointProps)]
    VDS_REPARSE_POINT_PROP** ppReparsePointProps,
  [out] long* plNumberOfReparsePointProps
);
```

ppReparsePointProps: A pointer to an array of **VDS_REPARSE_POINT_PROP** structures that, if the operation is successfully completed, receives the array of reparse point properties.

plNumberOfReparsePointProps: A pointer to a variable that, if the operation is successfully completed, receives the total number of elements returned in **ppReparsePointPorps**.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **ppReparsePointProps** is not NULL.
- Verify that **plNumberOfReparsePointProps** is not NULL.

The server MUST point **ppReparsePointProps** to an array of **VDS_REPARSE_POINT_PROP** structures that contain information about each reparse point on the volume, point **plNumberOfReparsePointProps** to the size of the array, and return an HRESULT indicating failure or success.

3.3.5.2.26.6 IVdsVolumeMF::DeleteAccessPath (Opnum 8)

The **DeleteAccessPath** method removes the access path from the current volume.

```
HRESULT DeleteAccessPath(
  [in, max_is(MAX_PATH - 1), string]
    WCHAR* pwszPath,
  [in] long bForce
);
```

pwszPath: A Unicode string indicating the access path, for example, "C:\myfolder\mydocuments".

bForce: A Boolean that determines whether an access is deleted unconditionally, even if the volume is in use. This parameter is meaningful only when the access path is a drive letter.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **pwszPath** is not NULL.
- Verify that the access path that **pwszPath** specifies is an access point to the volume.

The server MUST then perform the following in sequence:

- Delete the access point from the volume.
- If the server determines that a mount point was removed from the volume, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_MOUNT_POINT.
  - **MountPoint** member is a **VDS_MOUNT_POINT_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_MOUNT_POINT_CHANGE.
    - **volumeId** is the **VDS_OBJECT_ID** of the volume object whose mount point was removed.
- If the server determines that a drive letter was removed from the volume, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_DRIVE_LETTER.
  - **Letter** member is a **VDS_DRIVE_LETTER_NOTIFICATION** that has the following attributes:
    - **ulEvent** is VDS_NF_DRIVE_LETTER_FREE.
    - **wcLetter** is the drive letter that was removed from the volume.
    - **volumeId** is the **VDS_OBJECT_ID** of the volume object whose drive letter was removed.
- Return an HRESULT indicating failure or success.

3.3.5.2.26.7 IVdsVolumeMF::Mount (Opnum 9)

The **Mount** method mounts a volume. This method has no parameters.

```
HRESULT Mount();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives the message, it MUST perform the following in sequence:

- If the volume was dismounted permanently, bring the volume online.
- Mount the volume.

- Return an HRESULT indicating failure or success.

3.3.5.2.26.8 IVdsVolumeMF::Dismount (Opnum 10)

The **Dismount** method dismounts a mounted volume.

```
HRESULT Dismount(
  [in] long bForce,
  [in] long bPermanent
);
```

bForce: A Boolean that determines whether the current volume is dismounted unconditionally, even if the volume is in use.

bPermanent: A Boolean that determines whether a volume MUST be dismounted permanently by taking the volume offline after dismounting it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following:

- Verify that the volume is not currently temporarily dismounted.
- If **bPermanent** is set, verify that the volume supports permanent dismount by checking the volume flag VDS_VF_PERMANENT_DISMOUNT_SUPPORTED.
- If **bPermanent** is set, verify that the volume does not have any access paths.
- If **bPermanent** is not set, verify that the volume is not currently permanently dismounted.

The server MUST then perform the following in sequence:

- Dismount the volume.
- If **bForce** is set, force the dismount, even if the volume is in use.
- If **bPermanent** is set, take the volume offline.
- Return an HRESULT indicating failure or success.

3.3.5.2.26.9 IVdsVolumeMF::SetFileSystemFlags (Opnum 11)

The **SetFileSystemFlags** method sets the file system flags.

```
HRESULT SetFileSystemFlags(
  [in] unsigned long ulFlags
);
```

ulFlags: Callers can set the VDS_FPF_COMPRESSED flag.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following:

- Verify that ulFlags does not contain any flags other than VDS_FPF_COMPRESSED.

The server MUST set the file system flags specified by **ulFlags** and return an HRESULT indicating failure or success.

3.3.5.2.26.10 IVdsVolumeMF::ClearFileSystemFlags (Opnum 12)

The **ClearFileSystemFlags** method clears the file system flags.

```
HRESULT ClearFileSystemFlags(
  [in] unsigned long ulFlags
);
```

ulFlags: Callers can clear the VDS_FPF_COMPRESSED flag.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ulFlags** does not contain any flags other than VDS_FPF_COMPRESSED.

The server MUST clear the file system flags that **ulFlags** specifies and return an HRESULT indicating failure or success.

3.3.5.2.27 IVdsVolumeMF2 Methods

3.3.5.2.27.1 IVdsVolumeMF2::GetFileSystemTypeName (Opnum 3)

The **GetFileSystemTypeName** method retrieves the name of the file system on a volume.<89>

```
HRESULT GetFileSystemTypeName(
  [out, string] WCHAR** ppwszFileSystemTypeName
);
```

ppwszFileSystemTypeName: A pointer that, if the operation is successfully completed, receives a null-terminated Unicode string with the file system name.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppwszFileSystemTypeName** is not NULL.

The server MUST point **ppwszFileSystemTypeName** to a string that contains the name of the file system on the volume, and then return an HRESULT indicating failure or success.

3.3.5.2.27.2 IVdsVolumeMF2::QueryFileSystemFormatSupport (Opnum 4)

The **QueryFileSystemFormatSupport** method retrieves the properties of the file systems that are supported for formatting a volume.<90>

```
HRESULT QueryFileSystemFormatSupport(
  [out, size_is(,*plNumberOfFileSystems)]
    VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP** ppFileSystemSupportProps,
  [out] long* plNumberOfFileSystems
);
```

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **ppFileSystemSupportProps** is not NULL.
- Verify that **plNumberOfFileSystems** is not NULL.

The server MUST point **ppFileSystemSupportProps** to an array of **VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP** structures that contain information about each file system that is supported for formatting the volume, point **plNumberOfFileSystems** to the size of the array, and return an HRESULT indicating failure or success.

3.3.5.2.27.3 IVdsVolumeMF2::FormatEx (Opnum 5)

The **FormatEx** method formats a file system on a volume.

```
HRESULT FormatEx(
  [in, unique, string] WCHAR* pwszFileSystemTypeName,
  [in] unsigned short usFileSystemRevision,
  [in] unsigned long ulDesiredUnitAllocationSize,
  [in, unique, string] WCHAR* pwszLabel,
  [in] long bForce,
  [in] long bQuickFormat,
  [in] long bEnableCompression,
  [out] IVdsAsync** ppAsync
);
```

pwszFileSystemTypeName: A null-terminated Unicode string that contains the name of the file systems to format the volume with.

usFileSystemRevision: A 16-bit, binary-coded decimal number that indicates the revision of the file system, if any. The first two (most significant) digits (8-bits) indicate the major revision, and the last two (least significant) digits (8-bits) indicate the minor revision.**Note** 0x0250 represents revision 2.50.

ulDesiredUnitAllocationSize: The size, in bytes, of the allocation unit for the file system. The value MUST be a power of 2. If the value is 0, a default allocation unit that is determined by the file system type is used. The allocation unit range is file system-dependent.

pwszLabel: A null-terminated Unicode string to assign to the new file system. The maximum label size is file system-dependent.

bForce: A Boolean that determines whether a file system format is forced, even if the partition is in use.

bQuickFormat: A Boolean that determines whether a file system is quick formatted. A quick format does not verify each sector on the volume.

bEnableCompression: A Boolean that determines whether a file system is created with compression enabled.<91>

ppAsync: A pointer to an **IVdsAsync** interface that, if the operation is successfully completed, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppAsync** is not NULL.

The server MUST perform the following:

- Create a new async object that implements the **IVdsAsync**interface with an output type of VDS_ASYNCOUT_FORMAT and then set the pointer that **ppAsync** references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Format the volume following the parameters specified to the method.
- For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - volume member is a **VDS_VOLUME_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the **VDS_OBJECT_ID** of this volume object.
- If TRUE is specified for **bEnableCompression**, and an NTFS file system is being formatted, compress the file system after formatting is complete.
- Set the return code in the async object to an HRESULT indicating failure or success.
  - If the server tried to compress the file systems after formatting but failed, the return code MUST be set to VDS_S_VOLUME_COMPRESS_FAILED (HRESULT of 0x00042443).
- If the task completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.28 IVdsVolumeShrink Methods

3.3.5.2.28.1 IVdsVolumeShrink::QueryMaxReclaimableBytes (Opnum 3)

The **QueryMaxReclaimableBytes** method retrieves the maximum number of bytes that can be reclaimed from the current volume.<92>

```
HRESULT QueryMaxReclaimableBytes(
  [out] ULONGLONG* pullMaxNumberOfReclaimableBytes
);
```

pullMaxNumberOfReclaimableBytes: A pointer to a variable that, if the operation is successfully completed, receives the maximum number of bytes that can be reclaimed from the current volume. This number is always a multiple of the file system **cluster size**, which is in turn a multiple of the disk sector size.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **pullMaxNumberOfReclaimableBytes** is not NULL.

The server MUST set values that **pullMaxNumberOfReclaimableBytes** references with the maximum number of bytes that can be reclaimed from the volume, and then return an HRESULT indicating failure or success.

3.3.5.2.28.2 IVdsVolumeShrink::Shrink (Opnum 4)

The **Shrink** method shrinks the volume and all plexes, and then returns the released extents. The **Shrink** method compacts the files toward the beginning of the volume, creating free space at the end of the volume. The **Shrink** method also truncates the file system, reducing its size, and then truncates the partition or dynamic volume.<93>

```
HRESULT Shrink(
  [in] ULONGLONG ullDesiredNumberOfReclaimableBytes,
  [in] ULONGLONG ullMinNumberOfReclaimableBytes,
  [out] IVdsAsync** ppAsync
);
```

ullDesiredNumberOfReclaimableBytes: The desired number of bytes to be reclaimed from the volume. The method SHOULD attempt to reclaim the desired number of bytes as specified by this parameter. If it is unable to do so, it SHOULD attempt to reclaim a size smaller than **ullDesiredNumberOfReclaimableBytes** but greater than or equal to **ullMinNumberOfReclaimableBytes**. The actual number of bytes reclaimed is always a multiple of the file system cluster size, which is in turn a multiple of the disk sector size.

ullMinNumberOfReclaimableBytes: The minimum number of bytes to be reclaimed from the volume. If the method cannot reclaim at least the minimum number of bytes as specified by this parameter, the method MUST fail and MUST NOT reclaim any bytes.

ppAsync: A pointer to an **IVdsAsync** interface that, if the operation is successfully completed, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppAsync** is not NULL.<94>
- Verify that **ullDesiredNumberOfReclaimableBytes** is not zero.
- Verify that **ullDesiredNumberOfReclaimableBytes** is greater than or equal to **ullMinNumberOfReclaimableBytes**.

The server MUST perform the following:

- Create a new async object implementing the **IVdsAsync**interface with an output type of VDS_ASYNCOUT_SHRINKVOLUME and set the pointer **ppAsync** references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- If the file system on the volume is an NTFS file system, shrink the file system following the parameters specified to the method.
- Shrink the volume and all its plexes following the parameters specified to the method.
- Release the extents that have been reclaimed and mark them as free extents.
- For each callback object that is registered in the list of callback objects, call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure that has the following attributes:
  - **objectType** member is VDS_NTT_VOLUME.
  - **Volume** member is a **VDS_VOLUME_NOTIFICATION** with the following attributes:
    - **ulEvent** is VDS_NF_VOLUME_MODIFY.
    - **volumeId** is the **VDS_OBJECT_ID** of this volume object.
- Set the return code in the async object to an HRESULT indicating failure or success.
- If the task completed successfully, set the percentage-completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage-completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.5.2.29 IVdsVolumeOnline Methods

3.3.5.2.29.1 IVdsVolumeOnline::Online (Opnum 3)

The **Online** method brings the volume online.<95>

```
HRESULT Online();
```

This method has no parameters.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST bring the volume online, and then return an HRESULT indicating failure or success.

3.3.5.2.30 IVdsVolumePlex Methods

3.3.5.2.30.1 IVdsVolumePlex::GetProperties (Opnum 3)

The **GetProperties** method retrieves the properties of the volume plex that are represented by the object exposing this interface and method.

```
HRESULT GetProperties(
  [out] VDS_VOLUME_PLEX_PROP* pPlexProperties
);
```

pPlexProperties: A pointer to a **VDS_VOLUME_PLEX_PROP** structure that, if the operation is successfully completed, receives the properties of the volume plex.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **pPlexProperties** is not NULL.

The server MUST populate the **VDS_VOLUME_PLEX_PROP** structure that **pPlexProperties** references with the properties of the volume plex, and then return an HRESULT indicating failure or success. For information on the VDS_VOLUME_PLEX_PROP structure, see section 2.2.2.13.2.1.

3.3.5.2.30.2 IVdsVolumePlex::GetVolume (Opnum 4)

The **GetVolume** method retrieves the volume that the volume plex belongs to.

```
HRESULT GetVolume(
  [out] IVdsVolume** ppVolume
);
```

ppVolume: A pointer to an **IVdsVolume** interface that, if the operation is successfully completed, receives the **IVdsVolume** interface of the volume object that the volume plex belongs to. Callers MUST release the interface when they are done with it.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameter:

- Verify that **ppVolume** is not NULL.

The server MUST point **ppVolume** to an **IVdsVolume** interface of the volume object that the volume plex object's volume pointer refers. The server MUST then return an HRESULT indicating failure or success.

3.3.5.2.30.3 IVdsVolumePlex::QueryExtents (Opnum 5)

The **QueryExtents** method returns all extents for the current plex.

```
HRESULT QueryExtents(
  [out, size_is(,*plNumberOfExtents)]
    VDS_DISK_EXTENT** ppExtentArray,
  [out] long* plNumberOfExtents
);
```

ppExtentArray: A pointer to an array of **VDS_DISK_EXTENT** structures that, if the operation is successfully completed, receives the array of disk extent properties.

plNumberOfExtents: A pointer to a variable that, if the operation is successfully completed, receives the total number of elements in **ppExtentArray**.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

When the server receives this message, it MUST validate the following parameters:

- Verify that **ppExtentArray** is not NULL.
- Verify that **plNumberOfExtents** is not NULL.

The server MUST point **ppExtentArray** to an array of **VDS_DISK_EXTENT** structures that contain information about each disk extent on the volume plex, point **plNumberOfExtents** to the size of the array, and then return an HRESULT indicating failure or success.

For removable media drives, the server MUST set the value of **diskId** in the **VDS_DISK_EXTENT** structure to the **VDS_OBJECT_ID** of the drive associated with the plex object.

For a removable media drive with no media, the server MUST return a single extent of type VDS_DET_UNKNOWN with values of **ullOffset** and **ullSize** set to 0.

3.3.5.2.30.4 IVdsVolumePlex::Repair (Opnum 6)

The **Repair** method repairs a fault-tolerant volume plex by moving defective members to good disks. Only plexes that are RAID 5, striped with parity, can be repaired with this method.

```
HRESULT Repair(
  [in, size_is(lNumberOfDisks)] VDS_INPUT_DISK* pInputDiskArray,
  [in] long lNumberOfDisks,
  [out] IVdsAsync** ppAsync
```

```
);
```

pInputDiskArray: An array of **VDS_INPUT_DISK** structures that describe the replacement disks. Only **diskId** and **ullSize** SHOULD be specified in each **VDS_INPUT_DISK** element.<96>

lNumberOfDisks: The number of elements in **pInputDiskArray**. <97>

ppAsync: A pointer to an **IVdsAsync** interface that, if the operation is successfully completed, receives the **IVdsAsync** interface to monitor and control this operation. Callers MUST release the interface when they are done with it. If the **Wait** method is called on the interface, the interface returned in the **VDS_ASYNC_OUTPUT** structure MUST be released as well. For information on asynchronous tasks, see section 3.3.5.1.7.

Return Values: The method MUST return zero to indicate success, or an implementation-specific nonzero error code to indicate failure.

ERROR_SUCCESS (0x00000000)

When the server receives this message, it MUST validate the following parameters:

- Verify that the volume plex is RAID-5.
- Verify that **pInputDiskArray** is not NULL.
- Verify that **ppAsync** is not NULL.

If the volume resides on a basic disk, the server MUST return VDS_E_NOT_SUPPORTED (HRESULT of 0x80042400).

The server MUST perform the following:

- Create a new async object implementing the **IVdsAsync**interface with an output type of VDS_ASYNCOUT_REPAIRVOLUMEPLEX and set the pointer that **ppAsync** references to the interface.
- Return an HRESULT indicating failure or success.

The server MUST then perform the following in sequence:

- Repair the volume plex corresponding to the **VDS_OBJECT_ID** that **PlexId** specifies from this volume by moving defective members to good disks.
- Set the return code in the async object to an HRESULT indicating failure or success.
- If the task completed successfully, set the percentage completed value in the async object to 100.
- Set the signal state in the async object to TRUE.

At any point in the previous sequence—before the percentage-completed value in the async object is 100—the server MAY update the percentage-completed value.

3.3.6 Timer Events

The VDS Protocol uses no timer events.

3.3.7 Other Local Events

The server SHOULD track changes in the storage configuration of the computer. These changes may be due to hardware failures, the administrator changing the hardware configuration, or the administrator configuring storage objects by using tools such as disk, volume, pack, partition, drive letter, and file system arrivals, removals, and modifications.<98>

Note If the server does not track changes in storage configuration, clients MAY be unable to perform configuration operations such as the ones that are specified in this topic.

3.3.7.1 Disk Pack Arrival (Dynamic Disks)

When the server detects that a new disk pack on the system has dynamic disks, the server MUST add a disk pack object for it. For information on how to add the pack object, see section 3.3.5.1.1.

3.3.7.2 Disk Pack Removal (Dynamic Disks)

When the server detects that a disk pack with dynamic disks was removed from the system, the server MUST remove the corresponding pack object. For information on how to remove the pack object, see section 3.3.5.1.2.

3.3.7.3 Disk Arrival

When the server detects a new disk connected to the system, the server MUST add a disk object for it. For information on how to add a disk object, see section 3.3.5.1.3.

3.3.7.4 Disk Removal

When the server detects that a disk was disconnected from the system, the server MUST remove the corresponding disk object. For information on how to remove a disk pack object, see section 3.3.5.1.4.

3.3.7.5 Volume Arrival

When the server detects a new volume on the system, the server MUST add a volume object for it. For information on how to add a volume object, see section 3.3.5.1.5.

3.3.7.6 Volume Removal

When the server detects that a volume was removed from the system, the server MUST remove the corresponding volume object. For information on how to remove a disk pack object, see section 3.3.5.1.6.

3.3.7.7 File System Modification

When the server detects that a volume was formatted, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_FILE_SYSTEM.

- FileSystem member is a **VDS_FILE_SYSTEM_NOTIFICATION** with the following attributes:
  - ulEvent is VDS_NF_FILE_SYSTEM_MODIFY.
  - volumeId is the **VDS_OBJECT_ID** of the volume object whose file system was formatted.

3.3.7.8 Mount Point Change

When the server detects that a volume's mount point has changed, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_MOUNT_POINT.
- MountPoint member is a **VDS_MOUNT_POINT_NOTIFICATION** with the following attributes:
  - ulEvent is VDS_NF_MOUNT_POINT_CHANGE.
  - volumeId is the **VDS_OBJECT_ID** of the volume object whose mount point was changed.

3.3.7.9 Drive Letter Assignment

When the server detects that a drive letter is assigned to a volume, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_DRIVE_LETTER.
- Letter member is a **VDS_DRIVE_LETTER_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_DRIVE_LETTER_ASSIGN.
  - wcLetter is the drive letter that was assigned to the volume.
  - volumeId is the **VDS_OBJECT_ID** of the volume object whose drive letter was assigned.

3.3.7.10 Drive Letter Removal

When the server detects that a drive letter was removed from a volume, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure with the following attributes:

- objectType member is VDS_NTT_DRIVE_LETTER.
- Letter member is a **VDS_DRIVE_LETTER_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_DRIVE_LETTER_FREE.
  - wcLetter is the drive letter that was removed from the volume.
  - volumeId is the **VDS_OBJECT_ID** of the volume object whose drive letter was removed.

3.3.7.11 Media Arrival

When the server detects that media was inserted into a removable media drive, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure with the following attributes:

- objectType member is VDS_NTT_VOLUME.
- Volume member is a **VDS_VOLUME_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_VOLUME_MODIFY.
  - volumeId is the **VDS_OBJECT_ID** of the volume object corresponding to the removable media drive.

Then, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure with the following attributes:

- objectType member is VDS_NTT_DISK.
- Disk member is a **VDS_DISK_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_DRIVE_MODIFY.
  - diskId is the **VDS_OBJECT_ID** of the disk object corresponding to the removable media drive.

3.3.7.12 Media Removal

When the server detects that media was ejected from a removable media drive, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure with the following attributes:

- objectType member is VDS_NTT_VOLUME.
- Volume member is a **VDS_VOLUME_NOTIFICATION** with the following attributes:
  - ulEvent is VDS_NF_VOLUME_MODIFY.
  - volumeId is the **VDS_OBJECT_ID** of the volume object corresponding to the removable media drive.

Then, for each callback object that is registered in the list of callback objects, the server MUST call the callback object's **IVdsAdviseSink::OnNotify()** method by using a **VDS_NOTIFICATION** structure that has the following attributes:

- objectType member is VDS_NTT_DISK.
- Disk member is a **VDS_DISK_NOTIFICATION** that has the following attributes:
  - ulEvent is VDS_NF_DRIVE_MODIFY.
  - diskId is the **VDS_OBJECT_ID** of the disk object corresponding to the removable media drive.

4 Protocol Examples

The following sections provide examples of how a VDS Protocol client and server communicate in common scenarios.

A VDS client typically performs these operations in the following order:

1. Starts the VDS session.
2. Registers for notifications.
3. Receives notifications.
4. Enumerates the VDS object.
5. Retrieve properties of the VDS object.
6. Performs tasks.
7. Unregisters for notifications.
8. Ends the VDS session.

4.1 VDS Sessions

4.1.1 Starting Sessions

The following is an example of a client starting a VDS session by retrieving an instance of the VDS service object.

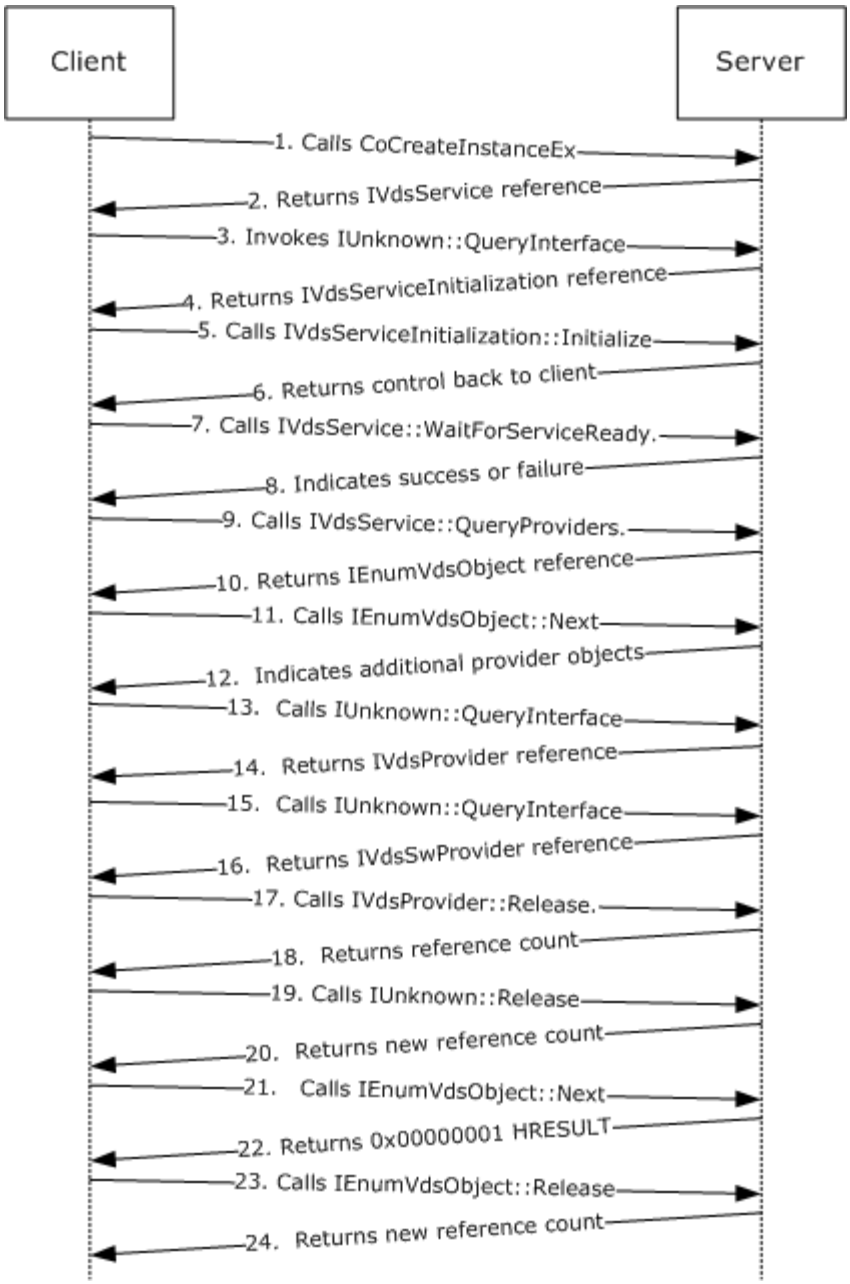

Figure 2: Client starting a VDS session by retrieving an instance of the VDS service object

1. The client requests the creation of a VDS session by calling **CoCreateInstanceEx** with the class GUID of the VDS service in order to create an instance of the VDS object on the server.

2. The server returns a reference to the **IVdsService** interface.

3. The client invokes the interface's **IUnknown::QueryInterface** method to request for the **IVdsServiceInitialization** interface.

4. The server returns a reference to the **IVdsServiceInitialization** interface.

5. The client calls the **IVdsServiceInitialization::Initialize** method.

6. The server begins initializing the service and returns control back to the client.

7. The client calls the **IVdsService::WaitForServiceReady** method.

8. The server replies to the client with an HRESULT indicating whether the service initialization was successful. If the VDS service initialization is successful (disk of 0x00000000), the client can request virtual disk management operations to the server through the methods in the **IVdsService** interface.

9. The client initiates the enumeration of providers by calling the **IVdsService::QueryProviders** method.

10. Upon successful execution of the **IVdsService::QueryProviders** method, the server creates an enumeration object and returns a reference to an **IEnumVdsObject** interface.

11. The client can call **IEnumVdsObject::Next** to retrieve the next provider in the enumeration.

12. Upon receiving the **IEnumVdsObject::Next** request, the server looks for the next provider object in the enumeration. If there is a provider object in the enumeration, the server returns an HRESULT of 0x00000000 and a reference to the IUnknown interface to the client. If the server reaches the end of the enumeration, the server returns a disk of 0x00000001.

13. If the server returns a zero disk, the client invokes the interface's **IUnknown::QueryInterface** method to request for the object's **IVdsProvider** interface.

14. The server returns a disk of 0x00000000 and a reference to the **IVdsProvider** interface to the client. The client may access the provider information through the **IVdsProvider** interface.

15. If the client wants to query the objects in the provider, the client invokes the interface's **IUnknown::QueryInterface** method to request for the object's **IVdsSwProvider** interface.

16. The server returns a disk of 0x00000000 and a reference to the **IVdsSwProvider** interface to the client. The client may enumerate the objects in the provider through the **IVdsProvider** interface.

17. When a client no longer needs the **IVdsProvider** interface, the client must release the reference to the interface by calling **IVdsProvider::Release**.

18. The server returns a new reference count for **IVdsProvider::Release**.

19. The client also needs to release the reference to the **IUnknown** interface by calling **IUnknown::Release**.

20. The server returns a new reference count for **IUnknown::Release**.

21. The client can call **IEnumVdsObject::Next** again for the next provider in the enumeration.

22. If the server reaches the end of the enumeration, the server returns a disk of 0x00000001.

23. The client no longer needs the **IEnumVdsObject** interface; therefore, it calls **IEnumVdsObject::Release** to release the reference.

24. The server returns a new reference count for **IEnumVdsObject**.

4.1.2 Ending Sessions

The following illustration shows an example of a client ending a VDS session.

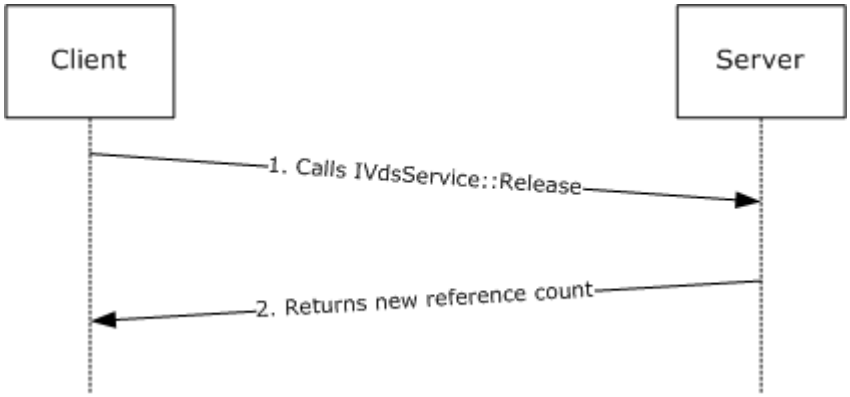

Figure 3: Client ending a VDS session

1. The client releases the reference to the **IVdsService** interface by invoking **IVdsService::Release**.

2. The server returns the new reference count for the **IVdsService** interface.

4.2 VDS Client Notifications

4.2.1 Registering for Notifications

The following illustration shows an example of a client that registers to receive notifications from a server.

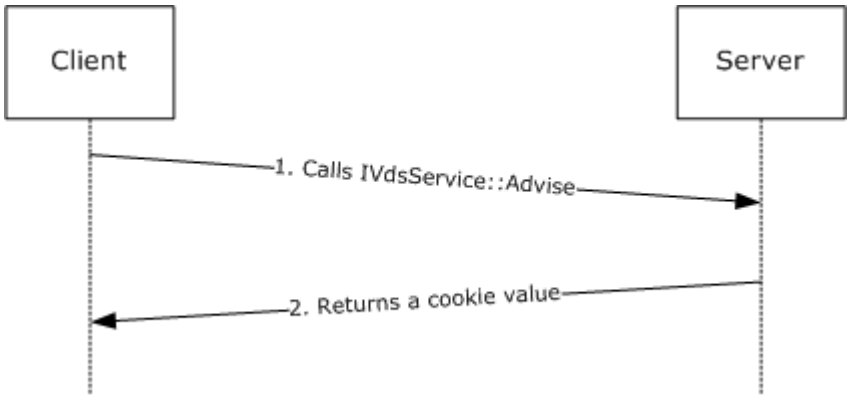

Figure 4: Client registering to receive notifications from server

1. The client requests registration by calling the **IVdsService::Advise** method and by passing an **IVdsAdviseSink** interface as a parameter.

2. The server returns a cookie value that uniquely identifies the client registration. The client can later use the cookie value to unregister for notifications.

4.2.2 Receiving Notifications

The following illustration shows an example of what happens when one or more VDS events are triggered.

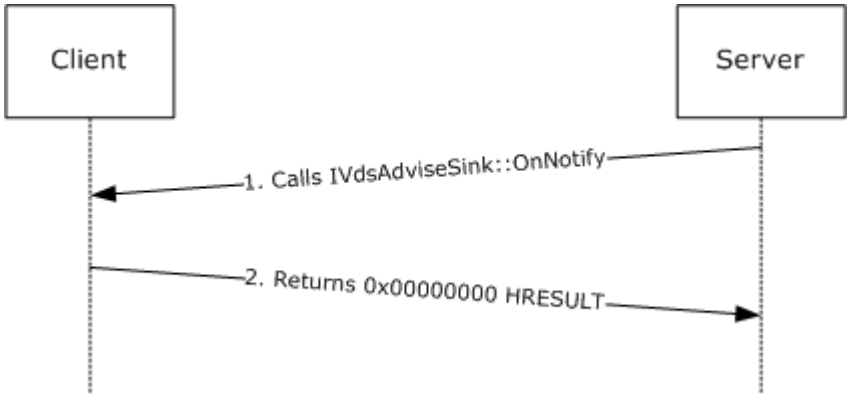

Figure 5: VDS event triggered

1. For each callback that was registered, the server calls **IVdsAdviseSink::OnNotify()** with an array of **VDS_NOTIFICATION** structures that describe the events that were triggered.

2. The client returns an HRESULT of 0x00000000 to acknowledge the notification.

4.2.3 Unregistering for Notifications

The following illustration shows an example of a client that cancels a previous registration for a notification.

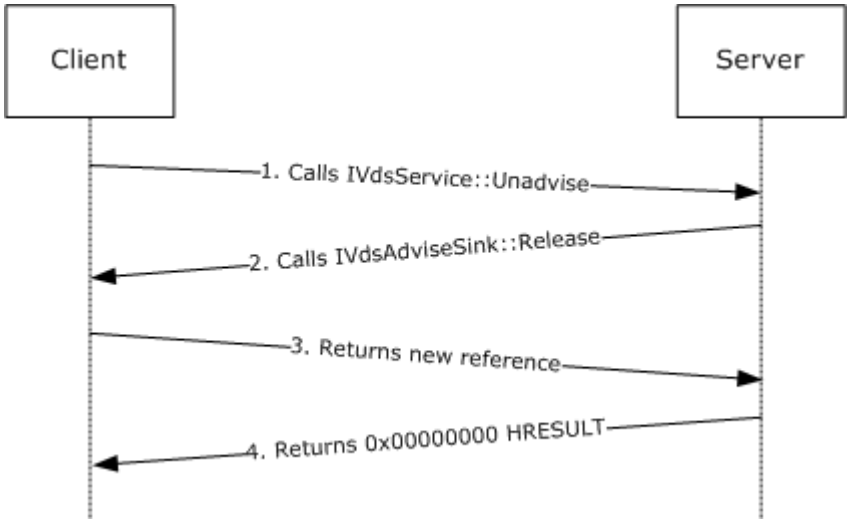

Figure 6: Client canceling previous registration for notification

1. The client requests unregistration by calling the **IVdsService::Unadvise** method and passing the cookie value that was received during registration.

2. The server determines that the cookie value matches a registered **IVdsAdviseSink** interface and invokes the **IVdsAdviseSink::Release** method to release its reference.

3. The client returns the new reference count for the **IVdsAdviseSink** interface.

4. The server returns an HRESULT of 0x00000000 in response to an **IVdsService::Unadvise** call from the client to acknowledge that the registration is canceled. The server may reuse the cookie value in the future.

4.3 Querying Enumerations of VDS Objects

Most VDS objects are retrievable only through an enumeration via the **IVdsPack** interface. The following illustration shows an example of a client enumerating volume objects belonging to a disk pack.

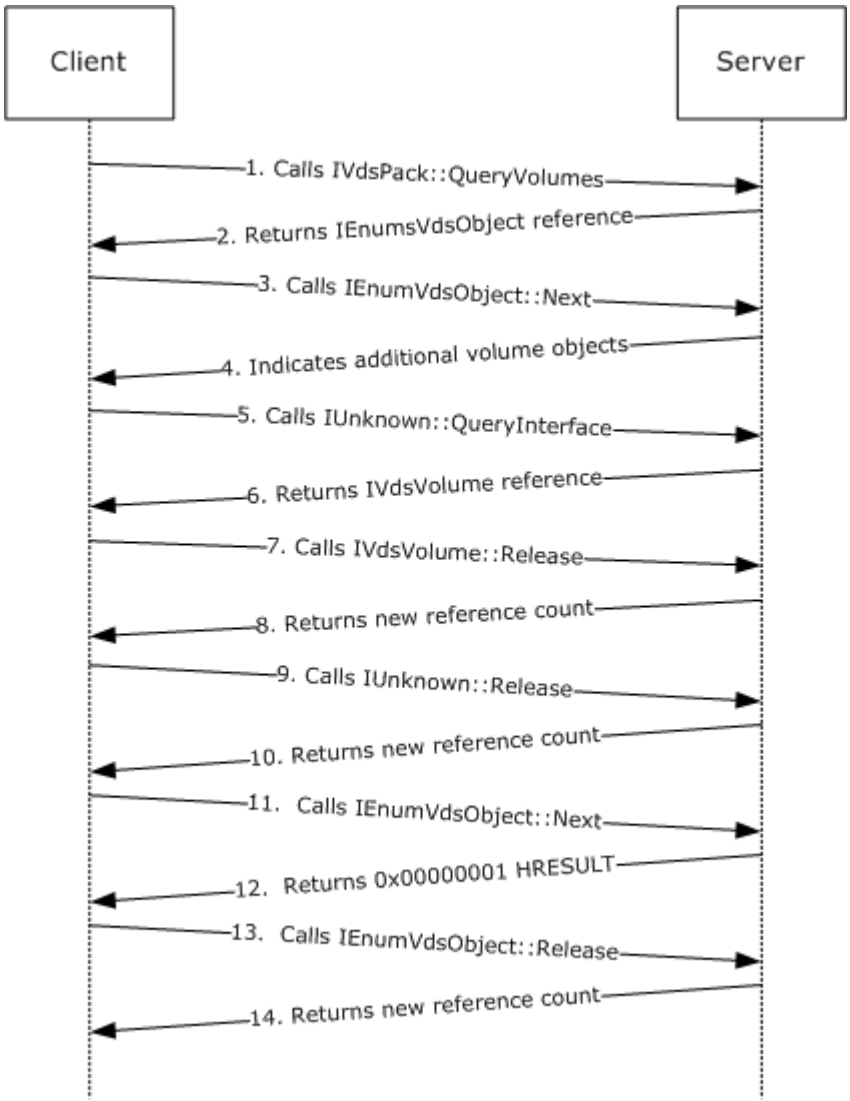

Figure 7: Client enumerating volume objects belonging to a disk pack

1. The client initiates the enumeration of volumes by calling the **IVdsPack::QueryVolumes** method.

2. Upon successful execution of the **IVdspack::QueryVolumes** method, the server creates an enumeration object and returns a reference to an **IEnumVdsObject** interface.

3. The client can call **IEnumVdsObject::Next** for the next object in the enumeration that it wants to retrieve.

4. Upon receiving the **IEnumVdsObject::Next** request, the server looks for the next volume object in the enumeration. If one exists, then the server returns an HRESULT of 0x00000000 and a reference to the **IUnknown** interface to the client. If the server reaches the end of the enumeration, the server returns an HRESULT of 0x00000001.

5. Assuming the server returned a zero HRESULT, the client invokes the interface's **IUnknown::QueryInterface** method to request for the object's **IVdsVolume** interface.

6. The server returns an HRESULT of 0x00000000 and a reference to the **IVdsVolume** interface to the client. The client may access the volume information through the **IVdsVolume** interface.

7. When a client no longer needs the **IVdsVolume**interface, the client must release the reference to the interface by calling **IVdsVolume::Release**.

8. The server returns a new reference count for **IVdsVolume::Release**.

9. The client also needs to release the reference to the **IUnknown** interface by calling **IUnknown::Release**.

10. The server returns a new reference count for **IUnknown::Release**.

11. The client can call **IEnumVDsObject::Next** again for the next object in the enumeration.

12. When the server reaches the end of the enumeration, the server returns an HRESULT of 0x00000001.

13. The client no longer needs the **IEnumVdsObject** interface, so it calls **IEnumVdsObject::Release** to release the reference.

14. The server returns a new reference count for **IEnumVdsObject**.

All other VDS objects that are retrievable via enumeration can be retrieved using similar steps.

4.4 Retrieving the Properties and IDs of VDS Objects

After an object is retrieved, a common task is to look for the VDS object ID, which uniquely identifies the object and is located in the object's properties structure.

The following illustration shows how to retrieve the properties of a volume object, if one exists. For information on how to retrieve a reference to a volume object, see section 4.3.

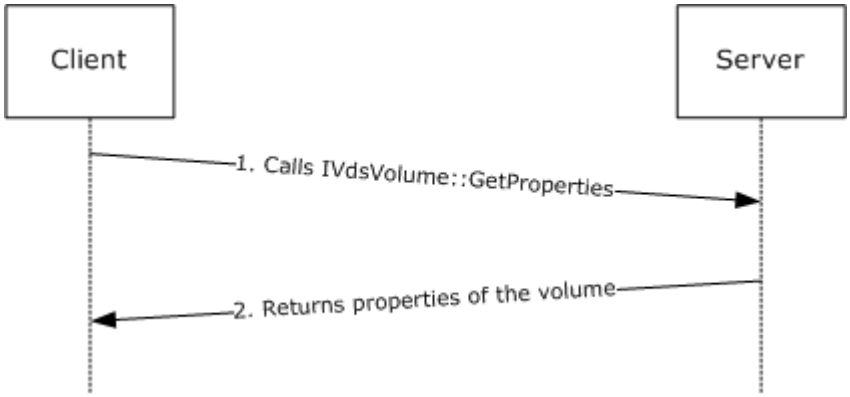

Figure 8: Retrieving the properties of a volume object

1. The client calls the **IVdsVolume::GetProperties** method, passing in a reference to a **VDS_VOLUME_PROP** structure in which to store the properties.

2. After successful execution of **IVdsVolume::GetProperties**, the server returns the properties of the volume, which includes its VDS object ID in the client-provided **VDS_VOLUME_PROP** structure.

After successful execution of the **IVdsVolume::GetProperties** request, which returns a filled **VDS_VOLUME_PROP** structure, the client can inspect any members of that structure.

The properties of other VDS objects can be retrieved by using similar steps.

4.5 Performing Asynchronous Tasks

The VDS Protocol exposes certain potentially long-running configuration tasks. Such tasks can be performed asynchronously. The following illustration shows an example of an asynchronous task, formatting a volume.

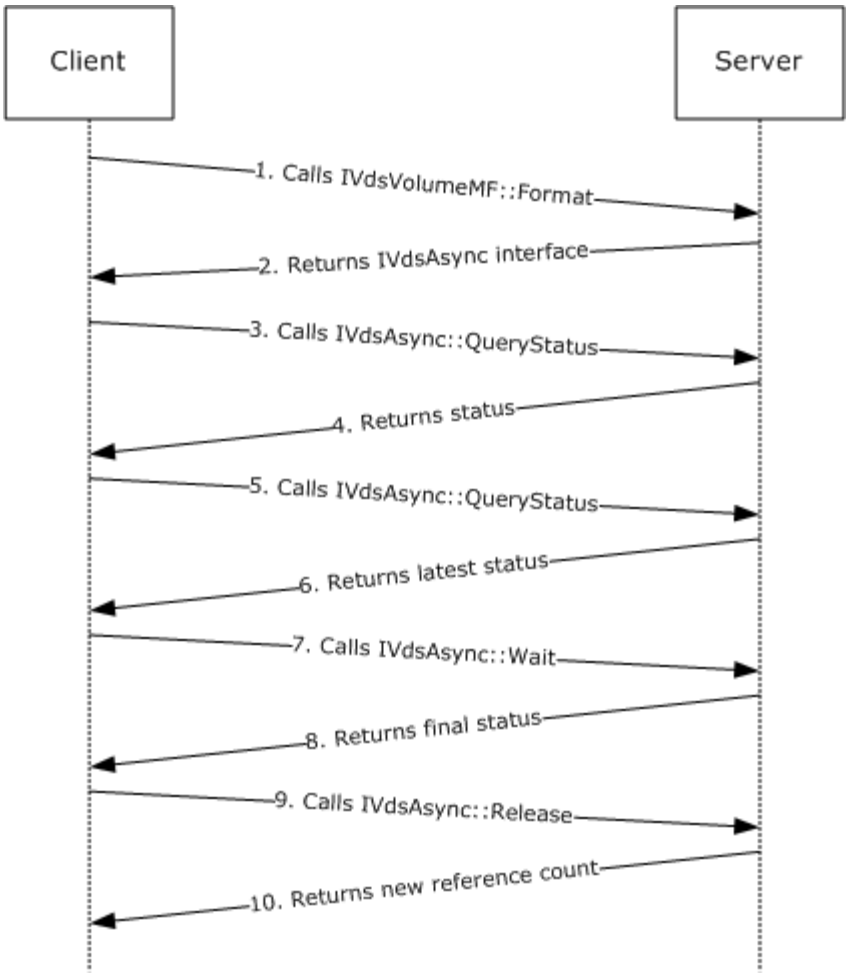

Figure 9: Asynchronous task of formatting a volume

1. The client requests that a volume be formatted by calling the **IVdsVolumeMF::Format** method.
2. The server acknowledges the format request and returns an **IVdsAsync** interface that the client can use to monitor progress of the format operation.
3. The client checks the current status of the format operation by calling the **IVdsAsync::QueryStatus** method on the returned **IVdsAsync** interface.
4. The server returns the status of the format operation.
5. The client can repeatedly check the status of the format operation by calling the **IVdsAsync::QueryStatus** method.
6. For each **IVdsAsync::QueryStatus** request, the server returns the latest status of the format operation.
7. The client can wait for the format operation to complete by calling the **IVdsAsync::Wait** method.

8. When the format operation completes, the server responds to the **IVdsAsync::Wait** call by returning the final status of the format.

9. The client invokes the **IVdsAsync::Release** method to release its reference.

10. The server returns the new reference count for the **IVdsAsync** interface.

5 Security

The following sections specify security considerations for implementers of the VDS Protocol.

5.1 Security Considerations for Implementers

The VDS Protocol introduces no security considerations except those that apply to DCOM Remote Protocol interfaces, as specified in [MS-DCOM] section 5.

5.2 Index of Security Parameters

This protocol has no security parameters.

6 Appendix A: Full IDL

For ease of implementation, the full **IDL** is provided here; "ms-dtyp.idl" is the IDL that is in the [MS-DTYP]Appendix A.

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

interface IEnumVdsObject;
interface IVdsAdviseSink;
interface IVdsAsync;
interface IVdsService;
interface IVdsServiceInitialization;
interface IVdsServiceUninstallDisk;
interface IVdsServiceHba;
interface IVdsServiceIscsi;
interface IVdsHbaPort;
interface IVdsIscsiInitiatorAdapter;
interface IVdsIscsiInitiatorPortal;
interface IVdsProvider;
interface IVdsSwProvider;
interface IVdsPack;
interface IVdsPack2;
interface IVdsDisk;
interface IVdsAdvancedDisk;
interface IVdsAdvancedDisk2;
interface IVdsCreatePartitionEx;
interface IVdsDiskPartitionMF;
interface IVdsRemovable;
interface IVdsVolume;
interface IVdsVolumeMF;
interface IVdsVolumeMF2;
interface IVdsVolumeShrink;
interface IVdsVolumeOnline;
interface IVdsVolumePlex;



#define MAX_PATH                    0x00000104

const unsigned long VER_VDS_LUN_INFORMATION     = 0x00000001;

typedef GUID    VDS_OBJECT_ID;

typedef enum _VDS_OBJECT_TYPE
{
    VDS_OT_UNKNOWN      = 0x00000000,
    VDS_OT_PROVIDER     = 0x00000001,
    VDS_OT_PACK         = 0x0000000A,
    VDS_OT_VOLUME       = 0x0000000B,
    VDS_OT_VOLUME_PLEX  = 0x0000000C,
    VDS_OT_DISK         = 0x0000000D,
    VDS_OT_HBAPORT      = 0x0000005A,
    VDS_OT_INIT_ADAPTER = 0x0000005B,
    VDS_OT_INIT_PORTAL  = 0x0000005C,
    VDS_OT_ASYNC        = 0x00000064,
```

```
    VDS_OT_ENUM         = 0x00000065
} VDS_OBJECT_TYPE;

typedef enum _VDS_SERVICE_FLAG
{
    VDS_SVF_SUPPORT_DYNAMIC             = 0x00000001,
    VDS_SVF_SUPPORT_FAULT_TOLERANT      = 0x00000002,
    VDS_SVF_SUPPORT_GPT                 = 0x00000004,
    VDS_SVF_SUPPORT_DYNAMIC_1394        = 0x00000008,
    VDS_SVF_CLUSTER_SERVICE_CONFIGURED  = 0x00000010,
    VDS_SVF_AUTO_MOUNT_OFF              = 0x00000020,
    VDS_SVF_OS_UNINSTALL_VALID          = 0x00000040,
    VDS_SVF_EFI                         = 0x00000080
} VDS_SERVICE_FLAG;

typedef enum _VDS_PROVIDER_TYPE
{
    VDS_PT_UNKNOWN  = 0x00000000,
    VDS_PT_SOFTWARE = 0x00000001,
    VDS_PT_HARDWARE = 0x00000002
} VDS_PROVIDER_TYPE;

typedef enum _VDS_PROVIDER_FLAG
{
    VDS_PF_DYNAMIC                          = 0x00000001,
    VDS_PF_ONE_DISK_ONLY_PER_PACK           = 0x00000004,
    VDS_PF_ONE_PACK_ONLINE_ONLY             = 0x00000008,
    VDS_PF_VOLUME_SPACE_MUST_BE_CONTIGUOUS  = 0x00000010,
    VDS_PF_SUPPORT_DYNAMIC_1394             = 0x20000000,
    VDS_PF_SUPPORT_FAULT_TOLERANT           = 0x40000000,
    VDS_PF_SUPPORT_DYNAMIC                  = 0x80000000
} VDS_PROVIDER_FLAG;

typedef enum _VDS_QUERY_PROVIDER_FLAG
{
    VDS_QUERY_SOFTWARE_PROVIDERS    = 0x00000001,
    VDS_QUERY_HARDWARE_PROVIDERS    = 0x00000002
} VDS_QUERY_PROVIDER_FLAG;

typedef enum _VDS_NOTIFICATION_TARGET_TYPE
{
    VDS_NTT_UNKNOWN         = 0x00000000,
    VDS_NTT_PACK            = 0x0000000A,
    VDS_NTT_VOLUME          = 0x0000000B,
    VDS_NTT_DISK            = 0x0000000D,
    VDS_NTT_PARTITION       = 0x0000003C,
    VDS_NTT_DRIVE_LETTER    = 0x0000003D,
    VDS_NTT_FILE_SYSTEM     = 0x0000003E,
    VDS_NTT_MOUNT_POINT     = 0x0000003F
} VDS_NOTIFICATION_TARGET_TYPE;

const unsigned long VDS_NF_PACK_ARRIVE              = 0x00000001;
const unsigned long VDS_NF_PACK_DEPART              = 0x00000002;
const unsigned long VDS_NF_PACK_MODIFY              = 0x00000003;

const unsigned long VDS_NF_VOLUME_ARRIVE            = 0x00000004;
const unsigned long VDS_NF_VOLUME_DEPART            = 0x00000005;
const unsigned long VDS_NF_VOLUME_MODIFY            = 0x00000006;
```

```
const unsigned long VDS_NF_VOLUME_REBUILDING_PROGRESS
                                                    = 0x00000007;

const unsigned long VDS_NF_DISK_ARRIVE              = 0x00000008;
const unsigned long VDS_NF_DISK_DEPART              = 0x00000009;
const unsigned long VDS_NF_DISK_MODIFY              = 0x0000000A;

const unsigned long VDS_NF_PARTITION_ARRIVE         = 0x0000000B;
const unsigned long VDS_NF_PARTITION_DEPART         = 0x0000000C;
const unsigned long VDS_NF_PARTITION_MODIFY         = 0x0000000D;

const unsigned long VDS_NF_DRIVE_LETTER_FREE        = 0x000000CC;
const unsigned long VDS_NF_DRIVE_LETTER_ASSIGN      = 0x000000CD;

const unsigned long VDS_NF_FILE_SYSTEM_MODIFY       = 0x000000CE;
const unsigned long VDS_NF_FILE_SYSTEM_FORMAT_PROGRESS
                                                    = 0x000000CF;

const unsigned long VDS_NF_MOUNT_POINTS_CHANGE      = 0x000000D0;

const unsigned long VDS_NF_FILE_SYSTEM_SHRINKING_PROGRESS
                                                    = 0x000000D1;

typedef struct _VDS_PACK_NOTIFICATION
{
    unsigned long   ulEvent;
    VDS_OBJECT_ID   packId;
} VDS_PACK_NOTIFICATION;

typedef struct _VDS_DISK_NOTIFICATION
{
    unsigned long   ulEvent;
    VDS_OBJECT_ID   diskId;
} VDS_DISK_NOTIFICATION;

typedef struct _VDS_VOLUME_NOTIFICATION
{
    unsigned long   ulEvent;
    VDS_OBJECT_ID   volumeId;
    VDS_OBJECT_ID   plexId;
    unsigned long   ulPercentCompleted;
} VDS_VOLUME_NOTIFICATION;

typedef struct _VDS_PARTITION_NOTIFICATION
{
    unsigned long   ulEvent;
    VDS_OBJECT_ID   diskId;
    ULONGLONG       ullOffset;
} VDS_PARTITION_NOTIFICATION;

typedef struct _VDS_DRIVE_LETTER_NOTIFICATION
{
    unsigned long   ulEvent;
    WCHAR           wcLetter;
    VDS_OBJECT_ID   volumeId;
} VDS_DRIVE_LETTER_NOTIFICATION;

typedef struct _VDS_FILE_SYSTEM_NOTIFICATION
```

```
{
    unsigned long   ulEvent;
    VDS_OBJECT_ID   volumeId;
    DWORD           dwPercentCompleted;
} VDS_FILE_SYSTEM_NOTIFICATION;

typedef struct _VDS_MOUNT_POINT_NOTIFICATION
{
    unsigned long   ulEvent;
    VDS_OBJECT_ID   volumeId;
} VDS_MOUNT_POINT_NOTIFICATION;

typedef struct _VDS_NOTIFICATION
{
    VDS_NOTIFICATION_TARGET_TYPE        objectType;
    [switch_is(objectType)] union
    {
    [case(VDS_NTT_PACK)]
        VDS_PACK_NOTIFICATION           Pack;
    [case(VDS_NTT_DISK)]
        VDS_DISK_NOTIFICATION           Disk;
    [case(VDS_NTT_VOLUME)]
        VDS_VOLUME_NOTIFICATION         Volume;
    [case(VDS_NTT_PARTITION)]
        VDS_PARTITION_NOTIFICATION      Partition;

    [case(VDS_NTT_DRIVE_LETTER)]
        VDS_DRIVE_LETTER_NOTIFICATION   Letter;
    [case(VDS_NTT_FILE_SYSTEM)]
        VDS_FILE_SYSTEM_NOTIFICATION    FileSystem;
    [case(VDS_NTT_MOUNT_POINT)]
        VDS_MOUNT_POINT_NOTIFICATION    MountPoint;

    [default];
    };
} VDS_NOTIFICATION;

typedef enum _VDS_ASYNC_OUTPUT_TYPE
{
    VDS_ASYNCOUT_UNKNOWN            = 0x00000000,
    VDS_ASYNCOUT_CREATEVOLUME       = 0x00000001,
    VDS_ASYNCOUT_EXTENDVOLUME       = 0x00000002,
    VDS_ASYNCOUT_SHRINKVOLUME       = 0x00000003,
    VDS_ASYNCOUT_ADDVOLUMEPLEX      = 0x00000004,
    VDS_ASYNCOUT_BREAKVOLUMEPLEX    = 0x00000005,
    VDS_ASYNCOUT_REMOVEVOLUMEPLEX   = 0x00000006,
    VDS_ASYNCOUT_REPAIRVOLUMEPLEX   = 0x00000007,
    VDS_ASYNCOUT_RECOVERPACK        = 0x00000008,
    VDS_ASYNCOUT_REPLACEDISK        = 0x00000009,
    VDS_ASYNCOUT_CREATEPARTITION    = 0x0000000A,
    VDS_ASYNCOUT_CLEAN              = 0x0000000B,
    VDS_ASYNCOUT_FORMAT             = 0x00000065
} VDS_ASYNC_OUTPUT_TYPE;

typedef struct _VDS_ASYNC_OUTPUT
{
    VDS_ASYNC_OUTPUT_TYPE type;
```

```
    [switch_is(type)] union
    {
        [case(VDS_ASYNCOUT_CREATEPARTITION)]
        struct _cp
        {
            ULONGLONG       ullOffset;
            VDS_OBJECT_ID   volumeId;
        } cp;

        [case(VDS_ASYNCOUT_CREATEVOLUME)]
        struct _cv
        {
            IUnknown        *pVolumeUnk;
        } cv;

        [case(VDS_ASYNCOUT_BREAKVOLUMEPLEX)]
        struct _bvp
        {
            IUnknown        *pVolumeUnk;
        } bvp;

        [case(VDS_ASYNCOUT_SHRINKVOLUME)]
        struct _sv
        {
            ULONGLONG       ullReclaimedBytes;
        } sv;

        [default];
    };
} VDS_ASYNC_OUTPUT;

typedef enum _VDS_HEALTH
{
    VDS_H_UNKNOWN                   = 0x00000000,
    VDS_H_HEALTHY                   = 0x00000001,
    VDS_H_REBUILDING                = 0x00000002,
    VDS_H_STALE                     = 0x00000003,
    VDS_H_FAILING                   = 0x00000004,
    VDS_H_FAILING_REDUNDANCY        = 0x00000005,
    VDS_H_FAILED_REDUNDANCY         = 0x00000006,
    VDS_H_FAILED_REDUNDANCY_FAILING = 0x00000007,
    VDS_H_FAILED                    = 0x00000008
} VDS_HEALTH;

typedef enum _VDS_TRANSITION_STATE
{
    VDS_TS_UNKNOWN      = 0x00000000,
    VDS_TS_STABLE       = 0x00000001,
    VDS_TS_EXTENDING    = 0x00000002,
    VDS_TS_SHRINKING    = 0x00000003,
    VDS_TS_RECONFIGING  = 0x00000004
} VDS_TRANSITION_STATE;

typedef enum _VDS_DRIVE_LETTER_FLAG
{
    VDS_DLF_NON_PERSISTENT  = 0x00000001
} VDS_DRIVE_LETTER_FLAG;
```

```
typedef enum _VDS_FILE_SYSTEM_TYPE
{
    VDS_FST_UNKNOWN = 0x00000000,
    VDS_FST_RAW     = 0x00000001,
    VDS_FST_FAT     = 0x00000002,
    VDS_FST_FAT32   = 0x00000003,
    VDS_FST_NTFS    = 0x00000004,
    VDS_FST_CDFS    = 0x00000005,
    VDS_FST_UDF     = 0x00000006
} VDS_FILE_SYSTEM_TYPE;

typedef enum _VDS_FILE_SYSTEM_FLAG
{
    VDS_FSF_SUPPORT_FORMAT          = 0x00000001,
    VDS_FSF_SUPPORT_QUICK_FORMAT    = 0x00000002,
    VDS_FSF_SUPPORT_COMPRESS        = 0x00000004,
    VDS_FSF_SUPPORT_SPECIFY_LABEL   = 0x00000008,
    VDS_FSF_SUPPORT_MOUNT_POINT     = 0x00000010,
    VDS_FSF_SUPPORT_REMOVABLE_MEDIA = 0x00000020,
    VDS_FSF_SUPPORT_EXTEND          = 0x00000040,
    VDS_FSF_ALLOCATION_UNIT_512     = 0x00010000,
    VDS_FSF_ALLOCATION_UNIT_1K      = 0x00020000,
    VDS_FSF_ALLOCATION_UNIT_2K      = 0x00040000,
    VDS_FSF_ALLOCATION_UNIT_4K      = 0x00080000,
    VDS_FSF_ALLOCATION_UNIT_8K      = 0x00100000,
    VDS_FSF_ALLOCATION_UNIT_16K     = 0x00200000,
    VDS_FSF_ALLOCATION_UNIT_32K     = 0x00400000,
    VDS_FSF_ALLOCATION_UNIT_64K     = 0x00800000,
    VDS_FSF_ALLOCATION_UNIT_128K    = 0x01000000,
    VDS_FSF_ALLOCATION_UNIT_256K    = 0x02000000
} VDS_FILE_SYSTEM_FLAG;

typedef enum _VDS_FILE_SYSTEM_FORMAT_SUPPORT_FLAG
{
    VDS_FSS_DEFAULT             = 0x00000001,
    VDS_FSS_PREVIOUS_REVISION   = 0x00000002,
    VDS_FSS_RECOMMENDED         = 0x00000004
} VDS_FILE_SYSTEM_FORMAT_SUPPORT_FLAG;

typedef enum _VDS_FILE_SYSTEM_PROP_FLAG
{
    VDS_FPF_COMPRESSED  = 0x00000001
} VDS_FILE_SYSTEM_PROP_FLAG;

typedef enum _VDS_PACK_STATUS
{
    VDS_PS_UNKNOWN  = 0x00000000,
    VDS_PS_ONLINE   = 0x00000001,
    VDS_PS_OFFLINE  = 0x00000004
} VDS_PACK_STATUS;

typedef enum _VDS_PACK_FLAG
{
    VDS_PKF_FOREIGN         = 0x00000001,
    VDS_PKF_NOQUORUM        = 0x00000002,
    VDS_PKF_POLICY          = 0x00000004,
    VDS_PKF_CORRUPTED       = 0x00000008,
    VDS_PKF_ONLINE_ERROR    = 0x00000010
```

```
} VDS_PACK_FLAG;

typedef enum _VDS_DISK_STATUS
{
    VDS_DS_UNKNOWN      = 0x00000000,
    VDS_DS_ONLINE       = 0x00000001,
    VDS_DS_NOT_READY    = 0x00000002,
    VDS_DS_NO_MEDIA     = 0x00000003,
    VDS_DS_FAILED       = 0x00000005,
    VDS_DS_MISSING      = 0x00000006
} VDS_DISK_STATUS;

typedef enum _VDS_DISK_FLAG
{
    VDS_DF_AUDIO_CD             = 0x00000001,
    VDS_DF_HOTSPARE             = 0x00000002,
    VDS_DF_RESERVE_CAPABLE      = 0x00000004,
    VDS_DF_MASKED               = 0x00000008,
    VDS_DF_STYLE_CONVERTIBLE    = 0x00000010,
    VDS_DF_CLUSTERED            = 0x00000020
} VDS_DISK_FLAG;

typedef enum _VDS_PARTITION_STYLE
{
    VDS_PST_UNKNOWN = 0x00000000,
    VDS_PST_MBR     = 0x00000001,
    VDS_PST_GPT     = 0x00000002
} VDS_PARTITION_STYLE;

typedef enum _VDS_PARTITION_FLAG
{
    VDS_PTF_SYSTEM  = 0x00000001
} VDS_PARTITION_FLAG;

typedef enum _VDS_LUN_RESERVE_MODE
{
    VDS_LRM_NONE            = 0x00000000,
    VDS_LRM_EXCLUSIVE_RW    = 0x00000001,
    VDS_LRM_EXCLUSIVE_RO    = 0x00000002,
    VDS_LRM_SHARED_RO       = 0x00000003,
    VDS_LRM_SHARED_RW       = 0x00000004
} VDS_LUN_RESERVE_MODE;

typedef enum _VDS_VOLUME_STATUS
{
    VDS_VS_UNKNOWN  = 0x00000000,
    VDS_VS_ONLINE   = 0x00000001,
    VDS_VS_NO_MEDIA = 0x00000003,
    VDS_VS_FAILED   = 0x00000005
} VDS_VOLUME_STATUS;

typedef enum _VDS_VOLUME_TYPE
{
    VDS_VT_UNKNOWN  = 0x00000000,
    VDS_VT_SIMPLE   = 0x0000000A,
    VDS_VT_SPAN     = 0x0000000B,
    VDS_VT_STRIPE   = 0x0000000C,
    VDS_VT_MIRROR   = 0x0000000D,
```

```
    VDS_VT_PARITY   = 0x0000000E
} VDS_VOLUME_TYPE;

typedef enum _VDS_VOLUME_FLAG
{
    VDS_VF_SYSTEM_VOLUME               = 0x00000001,
    VDS_VF_BOOT_VOLUME                 = 0x00000002,
    VDS_VF_ACTIVE                      = 0x00000004,
    VDS_VF_READONLY                    = 0x00000008,
    VDS_VF_HIDDEN                      = 0x00000010,
    VDS_VF_CAN_EXTEND                  = 0x00000020,
    VDS_VF_CAN_SHRINK                  = 0x00000040,
    VDS_VF_PAGEFILE                    = 0x00000080,
    VDS_VF_HIBERNATION                 = 0x00000100,
    VDS_VF_CRASHDUMP                   = 0x00000200,
    VDS_VF_INSTALLABLE                 = 0x00000400,
    VDS_VF_LBN_REMAP_ENABLED           = 0x00000800,
    VDS_VF_FORMATTING                  = 0x00001000,
    VDS_VF_NOT_FORMATTABLE             = 0x00002000,
    VDS_VF_NTFS_NOT_SUPPORTED          = 0x00004000,
    VDS_VF_FAT32_NOT_SUPPORTED         = 0x00008000,
    VDS_VF_FAT_NOT_SUPPORTED           = 0x00010000,
    VDS_VF_NO_DEFAULT_DRIVE_LETTER     = 0x00020000,
    VDS_VF_PERMANENTLY_DISMOUNTED      = 0x00040000,
    VDS_VF_PERMANENT_DISMOUNT_SUPPORTED = 0x00080000,
    VDS_VF_SHADOW_COPY                 = 0x00100000,
    VDS_VF_FVE_ENABLED                 = 0x00200000
} VDS_VOLUME_FLAG;

typedef enum _VDS_VOLUME_PLEX_TYPE
{
    VDS_VPT_UNKNOWN = 0x00000000,
    VDS_VPT_SIMPLE  = 0x0000000A,
    VDS_VPT_SPAN    = 0x0000000B,
    VDS_VPT_STRIPE  = 0x0000000C,
    VDS_VPT_PARITY  = 0x0000000E
} VDS_VOLUME_PLEX_TYPE;

typedef enum _VDS_VOLUME_PLEX_STATUS
{
    VDS_VPS_UNKNOWN     = 0x00000000,
    VDS_VPS_ONLINE      = 0x00000001,
    VDS_VPS_NO_MEDIA    = 0x00000003,
    VDS_VPS_FAILED      = 0x00000005
} VDS_VOLUME_PLEX_STATUS;

typedef enum _VDS_DISK_EXTENT_TYPE
{
    VDS_DET_UNKNOWN     = 0x00000000,
    VDS_DET_FREE        = 0x00000001,
    VDS_DET_DATA        = 0x00000002,
    VDS_DET_OEM         = 0x00000003,
    VDS_DET_ESP         = 0x00000004,
    VDS_DET_MSR         = 0x00000005,
    VDS_DET_LDM         = 0x00000006,
    VDS_DET_UNUSABLE    = 0x00007FFF
} VDS_DISK_EXTENT_TYPE;
```

```
typedef enum tag_VDS_PARTITION_STYLE
{
    VDS_PARTITION_STYLE_MBR = 0,
    VDS_PARTITION_STYLE_GPT = 1,
    VDS_PARTITION_STYLE_RAW = 2
} __VDS_PARTITION_STYLE;

typedef struct _VDS_PARTITION_INFO_MBR
{
    byte    partitionType;
    boolean bootIndicator;
    boolean recognizedPartition;
    DWORD   hiddenSectors;
} VDS_PARTITION_INFO_MBR;

typedef struct _VDS_PARTITION_INFO_GPT
{
    GUID        partitionType;
    GUID        partitionId;
    ULONGLONG   attributes;
    WCHAR       name[24];
} VDS_PARTITION_INFO_GPT;

typedef enum _VDS_STORAGE_IDENTIFIER_CODE_SET
{
    VDSStorageIdCodeSetReserved = 0x00000000,
    VDSStorageIdCodeSetBinary   = 0x00000001,
    VDSStorageIdCodeSetAscii    = 0x00000002,
    VDSStorageIdCodeSetUtf8     = 0x00000003
} VDS_STORAGE_IDENTIFIER_CODE_SET;

typedef enum _VDS_STORAGE_IDENTIFIER_TYPE
{
    VDSStorageIdTypeVendorSpecific              = 0x00000000,
    VDSStorageIdTypeVendorId                    = 0x00000001,
    VDSStorageIdTypeEUI64                       = 0x00000002,
    VDSStorageIdTypeFCPHName                    = 0x00000003,
    VDSStorageIdTypePortRelative                = 0x00000004,
    VDSStorageIdTypeTargetPortGroup             = 0x00000005,
    VDSStorageIdTypeLogicalUnitGroup            = 0x00000006,
    VDSStorageIdTypeMD5LogicalUnitIdentifier    = 0x00000007,
    VDSStorageIdTypeScsiNameString              = 0x00000008
} VDS_STORAGE_IDENTIFIER_TYPE;

typedef enum _VDS_STORAGE_BUS_TYPE
{
    VDSBusTypeUnknown       = 0x00000000,
    VDSBusTypeScsi          = 0x00000001,
    VDSBusTypeAtapi         = 0x00000002,
    VDSBusTypeAta           = 0x00000003,
    VDSBusType1394          = 0x00000004,
    VDSBusTypeSsa           = 0x00000005,
    VDSBusTypeFibre         = 0x00000006,
    VDSBusTypeUsb           = 0x00000007,
    VDSBusTypeRAID          = 0x00000008,
    VDSBusTypeiScsi         = 0x00000009,
    VDSBusTypeSas           = 0x0000000A,
    VDSBusTypeSata          = 0x0000000B,
```

```
    VDSBusTypeSd            = 0x0000000C,
    VDSBusTypeMmc           = 0x0000000D,
    VDSBusTypeMax           = 0x0000000E,
    VDSBusTypeMaxReserved   = 0x0000007F
} VDS_STORAGE_BUS_TYPE;

typedef struct _VDS_STORAGE_IDENTIFIER
{
    VDS_STORAGE_IDENTIFIER_CODE_SET m_CodeSet;
    VDS_STORAGE_IDENTIFIER_TYPE     m_Type;
    unsigned long                   m_cbIdentifier;
    [size_is(m_cbIdentifier)] byte  *m_rgbIdentifier;
} VDS_STORAGE_IDENTIFIER;

typedef struct _VDS_STORAGE_DEVICE_ID_DESCRIPTOR
{
    unsigned long                                   m_version;
    unsigned long                                   m_cIdentifiers;
    [size_is(m_cIdentifiers)] VDS_STORAGE_IDENTIFIER
        *m_rgIdentifiers;
} VDS_STORAGE_DEVICE_ID_DESCRIPTOR;

typedef enum _VDS_INTERCONNECT_ADDRESS_TYPE
{
    VDS_IA_UNKNOWN  = 0x00000000,
    VDS_IA_FCFS     = 0x00000001,
    VDS_IA_FCPH     = 0x00000002,
    VDS_IA_FCPH3    = 0x00000003,
    VDS_IA_MAC      = 0x00000004,
    VDS_IA_SCSI     = 0x00000005
} VDS_INTERCONNECT_ADDRESS_TYPE;

typedef struct _VDS_INTERCONNECT
{
    VDS_INTERCONNECT_ADDRESS_TYPE   m_addressType;
    unsigned long                   m_cbPort;
    [size_is(m_cbPort)] byte        *m_pbPort;
    unsigned long                   m_cbAddress;
    [size_is(m_cbAddress)] byte     *m_pbAddress;
} VDS_INTERCONNECT;

typedef struct _VDS_LUN_INFORMATION
{
    unsigned long                       m_version;
    byte                                m_DeviceType;
    byte                                m_DeviceTypeModifier;
    long                                m_bCommandQueuing;
    VDS_STORAGE_BUS_TYPE                m_BusType;
    [string] char                       * m_szVendorId;
    [string] char                       * m_szProductId;
    [string] char                       * m_szProductRevision;
    [string] char                       * m_szSerialNumber;
    GUID                                m_diskSignature;
    VDS_STORAGE_DEVICE_ID_DESCRIPTOR    m_deviceIdDescriptor;
    unsigned long                       m_cInterconnects;
    [size_is(m_cInterconnects)] VDS_INTERCONNECT *
        m_rgInterconnects;
} VDS_LUN_INFORMATION;
```

```
typedef enum VDS_IPADDRESS_TYPE
{
    VDS_IPT_TEXT    = 0x00000000,
    VDS_IPT_IPV4    = 0x00000001,
    VDS_IPT_IPV6    = 0x00000002,
    VDS_IPT_EMPTY   = 0x00000003
} VDS_IPADDRESS_TYPE;

typedef enum _VDS_HBAPORT_TYPE
{
    VDS_HPT_UNKNOWN     = 0x00000001,
    VDS_HPT_OTHER       = 0x00000002,
    VDS_HPT_NOTPRESENT  = 0x00000003,
    VDS_HPT_NPORT       = 0x00000005,
    VDS_HPT_NLPORT      = 0x00000006,
    VDS_HPT_FLPORT      = 0x00000007,
    VDS_HPT_FPORT       = 0x00000008,
    VDS_HPT_EPORT       = 0x00000009,
    VDS_HPT_GPORT       = 0x0000000A,
    VDS_HPT_LPORT       = 0x00000014,
    VDS_HPT_PTP         = 0x00000015
} VDS_HBAPORT_TYPE;

typedef enum _VDS_HBAPORT_STATUS
{
    VDS_HPS_UNKNOWN     = 0x00000001,
    VDS_HPS_ONLINE      = 0x00000002,
    VDS_HPS_OFFLINE     = 0x00000003,
    VDS_HPS_BYPASSED    = 0x00000004,
    VDS_HPS_DIAGNOSTICS = 0x00000005,
    VDS_HPS_LINKDOWN    = 0x00000006,
    VDS_HPS_ERROR       = 0x00000007,
    VDS_HPS_LOOPBACK    = 0x00000008
} VDS_HBAPORT_STATUS;

typedef enum _VDS_HBAPORT_SPEED_FLAG
{
    VDS_HSF_UNKNOWN         = 0x00000000,
    VDS_HSF_1GBIT           = 0x00000001,
    VDS_HSF_2GBIT           = 0x00000002,
    VDS_HSF_10GBIT          = 0x00000004,
    VDS_HSF_4GBIT           = 0x00000008,
    VDS_HSF_NOT_NEGOTIATED  = 0x00008000
} VDS_HBAPORT_SPEED_FLAG;

typedef enum _VDS_PATH_STATUS
{
    VDS_MPS_UNKNOWN     = 0x00000000,
    VDS_MPS_ONLINE      = 0x00000001,
    VDS_MPS_FAILED      = 0x00000005,
    VDS_MPS_STANDBY     = 0x00000007
} VDS_PATH_STATUS;

typedef struct VDS_REPARSE_POINT_PROP
{
    VDS_OBJECT_ID   SourceVolumeId;
    [string] WCHAR  *pwszPath;
```

```
} VDS_REPARSE_POINT_PROP, *PVDS_REPARSE_POINT_PROP;

typedef struct _VDS_DRIVE_LETTER_PROP
{
    WCHAR           wcLetter;
    VDS_OBJECT_ID   volumeId;
    unsigned long   ulFlags;
    long            bUsed;
} VDS_DRIVE_LETTER_PROP, *PVDS_DRIVE_LETTER_PROP;

typedef struct _VDS_FILE_SYSTEM_TYPE_PROP
{
    VDS_FILE_SYSTEM_TYPE    type;
    WCHAR                   wszName[8];
    unsigned long           ulFlags;
    unsigned long           ulCompressionFlags;
    unsigned long           ulMaxLabelLength;
    [string] WCHAR          *pwszIllegalLabelCharSet;
} VDS_FILE_SYSTEM_TYPE_PROP, *PVDS_FILE_SYSTEM_TYPE_PROP;

typedef struct _VDS_FILE_SYSTEM_PROP
{
    VDS_FILE_SYSTEM_TYPE    type;
    VDS_OBJECT_ID           volumeId;
    unsigned long           ulFlags;
    ULONGLONG               ullTotalAllocationUnits;
    ULONGLONG               ullAvailableAllocationUnits;
    unsigned long           ulAllocationUnitSize;
    [string] WCHAR          *pwszLabel;
} VDS_FILE_SYSTEM_PROP, *PVDS_FILE_SYSTEM_PROP;

typedef struct _VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP
{
    unsigned long               ulFlags;
    unsigned short              usRevision;
    unsigned long               ulDefaultUnitAllocationSize;
    unsigned long
 rgulAllowedUnitAllocationSizes[20];
    WCHAR                       wszName[20];
} VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP,
  *PVDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP;

typedef struct _VDS_DISK_EXTENT
{
    VDS_OBJECT_ID           diskId;
    VDS_DISK_EXTENT_TYPE    type;
    ULONGLONG               ullOffset;
    ULONGLONG               ullSize;
    VDS_OBJECT_ID           volumeId;
    VDS_OBJECT_ID           plexId;
    unsigned long           memberIdx;
} VDS_DISK_EXTENT, *PVDS_DISK_EXTENT;

typedef struct _VDS_PARTITION_PROP
{
    VDS_PARTITION_STYLE         PartitionStyle;
    unsigned long               ulFlags;
    unsigned long               ulPartitionNumber;
```

```
    ULONGLONG                   ullOffset;
    ULONGLONG                   ullSize;
    [switch_is(PartitionStyle)] union
    {
        [case(VDS_PST_MBR)]
        VDS_PARTITION_INFO_MBR  Mbr;

        [case(VDS_PST_GPT)]
        VDS_PARTITION_INFO_GPT  Gpt;

        [default];
    };
} VDS_PARTITION_PROP;

typedef struct _VDS_INPUT_DISK
{
    VDS_OBJECT_ID   diskId;
    ULONGLONG       ullSize;
    VDS_OBJECT_ID   plexId;
    unsigned long   memberIdx;
} VDS_INPUT_DISK;

typedef struct _CREATE_PARTITION_PARAMETERS
{
    VDS_PARTITION_STYLE style;
    [switch_is(style)] union
    {
        [case(VDS_PST_MBR)]
        struct
        {
            byte        partitionType;
            boolean     bootIndicator;
        } MbrPartInfo;

        [case(VDS_PST_GPT)]
        struct
        {
            GUID        partitionType;
            GUID        partitionId;
            ULONGLONG   attributes;
            WCHAR       name[24];
        } GptPartInfo;

        [default];
    };
} CREATE_PARTITION_PARAMETERS;

typedef struct _CHANGE_ATTRIBUTES_PARAMETERS
{
    VDS_PARTITION_STYLE style;   // legal values: MBR or GPT
    [switch_is(style)] union
    {
        [case(VDS_PST_MBR)]
        struct
        {
            boolean     bootIndicator;
        } MbrPartInfo;
```

```
        [case(VDS_PST_GPT)]
        struct
        {
            ULONGLONG   attributes;
        } GptPartInfo;

        [default];
    };
} CHANGE_ATTRIBUTES_PARAMETERS;

typedef struct _CHANGE_PARTITION_TYPE_PARAMETERS
{
    VDS_PARTITION_STYLE style;
    [switch_is(style)] union
    {
        [case(VDS_PST_MBR)]
        struct
        {
            byte        partitionType;
        } MbrPartInfo;

        [case(VDS_PST_GPT)]
        struct
        {
            GUID        partitionType;
        } GptPartInfo;

        [default];
    };

} CHANGE_PARTITION_TYPE_PARAMETERS;

typedef struct _VDS_WWN
{
    unsigned char   rguchWwn[8];
} VDS_WWN;

typedef struct _VDS_IPADDRESS
{
    VDS_IPADDRESS_TYPE  type;
    unsigned long       ipv4Address;
    unsigned char       ipv6Address[16];
    unsigned long       ulIpv6FlowInfo;
    unsigned long       ulIpv6ScopeId;
    WCHAR               wszTextAddress[256 + 1];
    unsigned long       ulPort;
} VDS_IPADDRESS;

typedef struct _VDS_ISCSI_SHARED_SECRET
{
    [size_is(ulSharedSecretSize)] unsigned char *
        pSharedSecret;
    unsigned long                           ulSharedSecretSize;
} VDS_ISCSI_SHARED_SECRET;

typedef struct _VDS_SERVICE_PROP
{
    [string] WCHAR  *pwszVersion;
```

```
    unsigned long   ulFlags;
} VDS_SERVICE_PROP;

typedef struct _VDS_HBAPORT_PROP
{
    VDS_OBJECT_ID       id;
    VDS_WWN             wwnNode;
    VDS_WWN             wwnPort;
    VDS_HBAPORT_TYPE    type;
    VDS_HBAPORT_STATUS  status;
    unsigned long       ulPortSpeed;
    unsigned long       ulSupportedPortSpeed;
} VDS_HBAPORT_PROP;

typedef struct _VDS_ISCSI_INITIATOR_ADAPTER_PROP
{
    VDS_OBJECT_ID   id;
    [string] WCHAR  *pwszName;
} VDS_ISCSI_INITIATOR_ADAPTER_PROP;

typedef struct _VDS_ISCSI_INITIATOR_PORTAL_PROP
{
    VDS_OBJECT_ID   id;
    VDS_IPADDRESS   address;
    unsigned long   ulPortIndex;
} VDS_ISCSI_INITIATOR_PORTAL_PROP;

typedef struct _VDS_PROVIDER_PROP
{
    VDS_OBJECT_ID       id;
    [string] WCHAR      *pwszName;
    GUID                guidVersionId;
    [string] WCHAR      *pwszVersion;
    VDS_PROVIDER_TYPE   type;
    unsigned long       ulFlags;
    unsigned long       ulStripeSizeFlags;
    short               sRebuildPriority;
} VDS_PROVIDER_PROP;

typedef struct _VDS_PACK_PROP
{
    VDS_OBJECT_ID   id;
    [string] WCHAR  *pwszName;
    VDS_PACK_STATUS status;
    unsigned long   ulFlags;
} VDS_PACK_PROP, *PVDS_PACK_PROP;

typedef struct _VDS_DISK_PROP
{
    VDS_OBJECT_ID           id;
    VDS_DISK_STATUS         status;
    VDS_LUN_RESERVE_MODE    ReserveMode;
    VDS_HEALTH              health;
    DWORD                   dwDeviceType;
    DWORD                   dwMediaType;
    ULONGLONG               ullSize;
    unsigned long           ulBytesPerSector;
    unsigned long           ulSectorsPerTrack;
```

```
    unsigned long           ulTracksPerCylinder;
    unsigned long           ulFlags;
    VDS_STORAGE_BUS_TYPE    BusType;
    VDS_PARTITION_STYLE     PartitionStyle;
    [switch_is(PartitionStyle)] union
    {
        [case(VDS_PST_MBR)]
        DWORD               dwSignature;

        [case(VDS_PST_GPT)]
        GUID                DiskGuid;

        [default];
    };
    [string] WCHAR          *pwszDiskAddress;
    [string] WCHAR          *pwszName;
    [string] WCHAR          *pwszFriendlyName;
    [string] WCHAR          *pwszAdaptorName;
    [string] WCHAR          *pwszDevicePath;
} VDS_DISK_PROP, *PVDS_DISK_PROP;

typedef struct _VDS_VOLUME_PROP
{
    VDS_OBJECT_ID           id;
    VDS_VOLUME_TYPE         type;
    VDS_VOLUME_STATUS       status;
    VDS_HEALTH              health;
    VDS_TRANSITION_STATE    TransitionState;
    ULONGLONG               ullSize;
    unsigned long           ulFlags;
    VDS_FILE_SYSTEM_TYPE    RecommendedFileSystemType;
    [string] WCHAR          *pwszName;
} VDS_VOLUME_PROP, *PVDS_VOLUME_PROP;

typedef struct _VDS_VOLUME_PLEX_PROP
{
    VDS_OBJECT_ID           id;
    VDS_VOLUME_PLEX_TYPE    type;
    VDS_VOLUME_PLEX_STATUS  status;
    VDS_HEALTH              health;
    VDS_TRANSITION_STATE    TransitionState;
    ULONGLONG               ullSize;
    unsigned long           ulStripeSize;
    unsigned long           ulNumberOfMembers;
} VDS_VOLUME_PLEX_PROP, *PVDS_VOLUME_PLEX_PROP;

[
    object,
    uuid(118610b7-8d94-4030-b5b8-500889788e4e),
    pointer_default(unique)
]
interface IEnumVdsObject : IUnknown
{
    [helpstring("method Next")]
    HRESULT Next(
                [in]  unsigned long celt,
                [out, size_is(celt), length_is(*pcFetched)]
                    IUnknown        **ppObjectArray,
```

```
                [out] unsigned long *pcFetched
            );

    [helpstring("method Skip")]
    HRESULT Skip(
                [in] unsigned long  celt
            );

    [helpstring("method Reset")]
    HRESULT Reset();

    [helpstring("method Clone")]
    HRESULT Clone(
                [out] IEnumVdsObject    **ppEnum
            );
}

[
    object,
    uuid(8326cd1d-cf59-4936-b786-5efc08798e25),
    pointer_default(unique)
]
interface IVdsAdviseSink : IUnknown
{
    [helpstring("method OnNotify")]
    HRESULT OnNotify(
           [in, range(1, 100)] long    lNumberOfNotifications,
           [in, size_is(lNumberOfNotifications)]
                VDS_NOTIFICATION       *pNotificationArray
            );
}

[
    object,
    uuid(d5d23b6d-5a55-4492-9889-397a3c2d2dbc),
    pointer_default(unique)
]
interface IVdsAsync : IUnknown
{
    [helpstring("method Cancel")]
    HRESULT Cancel();

    [helpstring("method Wait")]
    HRESULT Wait(
                [out] HRESULT           *pHrResult,
                [out] VDS_ASYNC_OUTPUT  *pAsyncOut
            );

    [helpstring("method QueryStatus")]
    HRESULT QueryStatus(
                [out] HRESULT       *pHrResult,
                [out] unsigned long *pulPercentCompleted
            );
}

[
    object,
    uuid(e0393303-90d4-4a97-ab71-e9b671ee2729),
```

```
    pointer_default(unique)
]

interface IVdsServiceLoader : IUnknown
{
    [helpstring("method LoadService")]
    HRESULT LoadService(
                [in,unique,string] LPWSTR       pwszMachineName,
                [out] IVdsService               **ppService
            );
}


[
    object,
    uuid(0818a8ef-9ba9-40d8-a6f9-e22833cc771e),
    pointer_default(unique)
]
interface IVdsService : IUnknown
{
    [helpstring("method IsServiceReady")]
    HRESULT IsServiceReady();

    [helpstring("method WaitForServiceReady")]
    HRESULT WaitForServiceReady();

    [helpstring("method GetProperties")]
    HRESULT GetProperties(
                [out] VDS_SERVICE_PROP  *pServiceProp
            );

    [helpstring("method QueryProviders")]
    HRESULT QueryProviders(
                [in] DWORD masks,
                [out] IEnumVdsObject    **ppEnum
            );

    HRESULT Opnum07NotUsedOnWire(void);

    [helpstring("method QueryUnallocatedDisks")]
    HRESULT QueryUnallocatedDisks(
                [out] IEnumVdsObject    **ppEnum
            );

    [helpstring("method GetObject")]
    HRESULT GetObject(
                [in] VDS_OBJECT_ID      ObjectId,
                [in] VDS_OBJECT_TYPE    type,
                [out] IUnknown          **ppObjectUnk
            );

    [helpstring("method QueryDriveLetters")]
    HRESULT QueryDriveLetters(
                [in] WCHAR                  wcFirstLetter,
                [in] DWORD                  count,
                [out, size_is(count)]
                   VDS_DRIVE_LETTER_PROP  *pDriveLetterPropArray
            );
```

```
    [helpstring("method QueryFileSystemTypes")]
    HRESULT QueryFileSystemTypes(
                [out, size_is(,*plNumberOfFileSystems)]
                      VDS_FILE_SYSTEM_TYPE_PROP
     **ppFileSystemTypeProps,
                [out] long                *plNumberOfFileSystems
            );

    [helpstring("method Reenumerate")]
    HRESULT Reenumerate();

    [helpstring("method Refresh")]
    HRESULT Refresh();

    [helpstring("method CleanupObsoleteMountPoints")]
    HRESULT CleanupObsoleteMountPoints();

    [helpstring("method Advise")]
    HRESULT Advise(
                [in] IVdsAdviseSink *pSink,
                [out] DWORD         *pdwCookie
            );

    [helpstring("method Unadvise")]
    HRESULT Unadvise(
                [in] DWORD  dwCookie
            );

    [helpstring("method Reboot")]
    HRESULT Reboot();

    [helpstring("method SetFlags")]
    HRESULT SetFlags(
                [in] unsigned long  ulFlags
            );

    [helpstring("method ClearFlags")]
    HRESULT ClearFlags(
                [in] unsigned long  ulFlags
            );
}

[
    object,
    uuid(4afc3636-db01-4052-80c3-03bbcb8d3c69),
    pointer_default(unique)
]
interface IVdsServiceInitialization : IUnknown
{
    [helpstring("method Initialize")]
    HRESULT Initialize(
                [in, unique, string] WCHAR  *pwszMachineName
            );
}

[
    object,
```

```
    uuid(B6B22DA8-F903-4be7-B492-C09D875AC9DA),
    pointer_default(unique)
]
interface IVdsServiceUninstallDisk : IUnknown
{
    [helpstring("method GetDiskIdFromLunInfo")]
    HRESULT GetDiskIdFromLunInfo(
                [in] VDS_LUN_INFORMATION    *pLunInfo,
                [out] VDS_OBJECT_ID         *pDiskId
            );

    [helpstring("method UninstallDisks")]
    HRESULT UninstallDisks(
                [in, size_is(ulCount)]
                    VDS_OBJECT_ID   *pDiskIdArray,
                [in] unsigned long  ulCount,
                [in] boolean        bForce,
                [out] boolean       *pbReboot,
                [out, size_is(ulCount)]
                    HRESULT         *pResults
            );
}

[
    object,
    uuid(0ac13689-3134-47c6-a17c-4669216801be),
    pointer_default(unique)
]
interface IVdsServiceHba : IUnknown
{
    [helpstring("method QueryHbaPorts")]
    HRESULT QueryHbaPorts(
                [out] IEnumVdsObject    **ppEnum
            );
}

[
    object,
    uuid(14fbe036-3ed7-4e10-90e9-a5ff991aff01),
    pointer_default(unique)
]
interface IVdsServiceIscsi : IUnknown
{
    [helpstring("method GetInitiatorName")]
    HRESULT GetInitiatorName(
                [out, string] WCHAR **ppwszIscsiName
            );

    [helpstring("method QueryInitiatorAdapters")]
    HRESULT QueryInitiatorAdapters(
                [out] IEnumVdsObject    **ppEnum
            );

    HRESULT Opnum05NotUsedOnWire(void);

    HRESULT Opnum06NotUsedOnWire(void);

    HRESULT Opnum07NotUsedOnWire(void);
```

```
    [helpstring("method SetInitiatorSharedSecret")]
    HRESULT SetInitiatorSharedSecret(
                [in, unique] VDS_ISCSI_SHARED_SECRET
     *pInitiatorSharedSecret,
                [in] VDS_OBJECT_ID                  targetId
            );

    HRESULT Opnum09NotUsedOnWire(void);
}

[
    object,
    uuid(2abd757f-2851-4997-9a13-47d2a885d6ca),
    pointer_default(unique)
]
interface IVdsHbaPort : IUnknown
{
    [helpstring("method GetProperties")]
    HRESULT GetProperties(
                [out] VDS_HBAPORT_PROP  *pHbaPortProp
            );

    [helpstring("method SetAllPathStatuses")]
    HRESULT SetAllPathStatuses(
                [in] VDS_PATH_STATUS    status
            );
}

[
    object,
    uuid(b07fedd4-1682-4440-9189-a39b55194dc5),
    pointer_default(unique)
]
interface IVdsIscsiInitiatorAdapter : IUnknown
{
    [helpstring("method GetProperties")]
    HRESULT GetProperties(
                [out] VDS_ISCSI_INITIATOR_ADAPTER_PROP
     *pInitiatorAdapterProp
            );

    [helpstring("method QueryInitiatorPortals")]
    HRESULT QueryInitiatorPortals(
                [out] IEnumVdsObject    **ppEnum
            );

    HRESULT Opnum05NotUsedOnWire(void);

    HRESULT Opnum06NotUsedOnWire(void);
}

[
    object,
    uuid(38a0a9ab-7cc8-4693-ac07-1f28bd03c3da),
    pointer_default(unique)
]
interface IVdsIscsiInitiatorPortal : IUnknown
```

```
{
    [helpstring("method GetProperties")]
    HRESULT GetProperties(
                [out] VDS_ISCSI_INITIATOR_PORTAL_PROP
     *pInitiatorPortalProp
            );

    [helpstring("method GetInitiatorAdapter")]
    HRESULT GetInitiatorAdapter(
                [out] IVdsIscsiInitiatorAdapter
     **ppInitiatorAdapter
            );

    HRESULT Opnum05NotUsedOnWire(void);

    HRESULT Opnum06NotUsedOnWire(void);

    HRESULT Opnum07NotUsedOnWire(void);

}

[
    object,
    uuid(10c5e575-7984-4e81-a56b-431f5f92ae42),
    pointer_default(unique)
]
interface IVdsProvider : IUnknown
{
    [helpstring("method GetProperties")]
    HRESULT GetProperties(
                [out] VDS_PROVIDER_PROP *pProviderProp
            );
}

[
    object,
    uuid(9aa58360-ce33-4f92-b658-ed24b14425b8),
    pointer_default(unique)
]
interface IVdsSwProvider : IUnknown
{
    [helpstring("method QueryPacks")]
    HRESULT QueryPacks(
                [out] IEnumVdsObject    **ppEnum
            );

    [helpstring("method CreatePack")]
    HRESULT CreatePack(
                [out] IVdsPack  **ppPack
            );
}

[
    object,
    uuid(d99bdaae-b13a-4178-9fdb-e27f16b4603e),
    pointer_default(unique)
]
```

```
interface IVdsHwProvider : IUnknown
{
    [helpstring("method QuerySubSystems")]
    HRESULT QuerySubSystems(
                [out] IEnumVdsObject **ppEnum
            );


}
[
    object,
    uuid(83bfb87f-43fb-4903-baa6-127f01029eec),
    pointer_default(unique)
]

interface IVdsSubSystemImportTarget : IUnknown
{
    [helpstring("method GetImportTarget")]
    HRESULT GetImportTarget(
                [out,string] LPWSTR *ppwszIscsiName
            );

    [helpstring("method SetImportTarget")]
    HRESULT SetImportTarget(
                [in,unique,string] LPWSTR pwszIscsiName
            );

}


[
    object,
    uuid(3b69d7f5-9d94-4648-91ca-79939ba263bf),
    pointer_default(unique)
]
interface IVdsPack : IUnknown
{
    [helpstring("method GetProperties")]
    HRESULT GetProperties(
                [out] VDS_PACK_PROP *pPackProp
            );

    [helpstring("method GetProvider")]
    HRESULT GetProvider(
                [out] IVdsProvider  **ppProvider
            );

    [helpstring("method QueryVolumes")]
    HRESULT QueryVolumes(
                [out] IEnumVdsObject    **ppEnum
            );

    [helpstring("method QueryDisks")]
    HRESULT QueryDisks(
                [out] IEnumVdsObject    **ppEnum
            );

    [helpstring("method CreateVolume")]
```

```
    HRESULT CreateVolume(
                [in] VDS_VOLUME_TYPE    type,
                [in, size_is(lNumberOfDisks)]
                     VDS_INPUT_DISK     *pInputDiskArray,
                [in] long               lNumberOfDisks,
                [in] unsigned long      ulStripeSize,
                [out] IVdsAsync         **ppAsync
            );

    [helpstring("method AddDisk")]
    HRESULT AddDisk(
                [in] VDS_OBJECT_ID          DiskId,
                [in] VDS_PARTITION_STYLE    PartitionStyle,
                [in] long                   bAsHotSpare
            );

    [helpstring("method MigrateDisks")]
    HRESULT MigrateDisks(
                [in, size_is(lNumberOfDisks)]
                     VDS_OBJECT_ID  *pDiskArray,
                [in] long lNumberOfDisks,
                [in] VDS_OBJECT_ID  TargetPack,
                [in] long           bForce,
                [in] long           bQueryOnly,
                [out, size_is(lNumberOfDisks)]
                     HRESULT        *pResults,
                [out] long          *pbRebootNeeded
            );

    HRESULT Opnum10NotUsedOnWire(void);

    [helpstring("method RemoveMissingDisk")]
    HRESULT RemoveMissingDisk(
                [in] VDS_OBJECT_ID  DiskId
            );

    [helpstring("method Recover")]
    HRESULT Recover(
                [out] IVdsAsync **ppAsync
            );
}

[
    object,
    uuid(13B50BFF-290A-47DD-8558-B7C58DB1A71A),
    pointer_default(unique)
]
interface IVdsPack2 : IUnknown
{
    [helpstring("method CreateVolume2")]
    HRESULT CreateVolume2(
                [in] VDS_VOLUME_TYPE    type,
                [in, size_is(lNumberOfDisks)]
                     VDS_INPUT_DISK     *pInputDiskArray,
                [in] long               lNumberOfDisks,
                [in] unsigned long      ulStripeSize,
                [in] unsigned long      ulAlign,
                [out] IVdsAsync         **ppAsync
```

```
            );
}

[
    object,
    uuid(07e5c822-f00c-47a1-8fce-b244da56fd06),
    pointer_default(unique)
]
interface IVdsDisk : IUnknown
{
    [helpstring("method GetProperties")]
    HRESULT GetProperties(
                [out] VDS_DISK_PROP *pDiskProperties
            );

    [helpstring("method GetPack")]
    HRESULT GetPack(
                [out] IVdsPack  **ppPack
            );

    [helpstring("method GetIdentificationData")]
    HRESULT GetIdentificationData(
                [out] VDS_LUN_INFORMATION   *pLunInfo
            );

    [helpstring("method QueryExtents")]
    HRESULT QueryExtents(
                [out, size_is(,*plNumberOfExtents)]
                      VDS_DISK_EXTENT   **ppExtentArray,
                [out] long              *plNumberOfExtents
            );

    [helpstring("method ConvertStyle")]
    HRESULT ConvertStyle(
                [in] VDS_PARTITION_STYLE    NewStyle
            );

    [helpstring("method SetFlags")]
    HRESULT SetFlags(
                [in] unsigned long  ulFlags
            );

    [helpstring("method ClearFlags")]
    HRESULT ClearFlags(
                [in] unsigned long  ulFlags
            );
}

[
    object,
    uuid(40F73C8B-687D-4a13-8D96-3D7F2E683936),
    pointer_default(unique)
]
interface IVdsDisk2 : IUnknown
{
    [helpstring("method SetSANMode")]
    HRESULT SetSANMode(
                [in] long   bEnable
```

```
            );
}

[
    object,
    uuid(6e6f6b40-977c-4069-bddd-ac710059f8c0),
    pointer_default(unique)
]
interface IVdsAdvancedDisk : IUnknown
{
    [helpstring("method GetPartitionProperties")]
    HRESULT GetPartitionProperties(
                [in] ULONGLONG              ullOffset,
                [out] VDS_PARTITION_PROP    *pPartitionProp
            );

    [helpstring("method QueryPartitions")]
    HRESULT QueryPartitions(
                [out, size_is(, *plNumberOfPartitions)]
                     VDS_PARTITION_PROP   **ppPartitionPropArray,
                [out] long                *plNumberOfPartitions
            );

    [helpstring("method CreatePartition")]
    HRESULT CreatePartition(
                [in] ULONGLONG                      ullOffset,
                [in] ULONGLONG                      ullSize,
                [in] CREATE_PARTITION_PARAMETERS    *para,
                [out] IVdsAsync                     **ppAsync
            );

    [helpstring("method DeletePartition")]
    HRESULT DeletePartition(
                [in] ULONGLONG  ullOffset,
                [in] long       bForce,
                [in] long       bForceProtected
            );

    [helpstring("method ChangeAttributes")]
    HRESULT ChangeAttributes(
                [in] ULONGLONG                      ullOffset,
                [in] CHANGE_ATTRIBUTES_PARAMETERS   *para
            );

    [helpstring("method AssignDriveLetter")]
    HRESULT AssignDriveLetter(
                [in] ULONGLONG  ullOffset,
                [in] WCHAR      wcLetter
            );

    [helpstring("method DeleteDriveLetter")]
    HRESULT DeleteDriveLetter(
                [in] ULONGLONG  ullOffset,
                [in] WCHAR      wcLetter
            );

    [helpstring("method GetDriveLetter")]
    HRESULT GetDriveLetter(
```

```
                [in] ULONGLONG  ullOffset,
                [out] WCHAR     *pwcLetter
            );

        [helpstring("method FormatPartition")]
        HRESULT FormatPartition(
                [in] ULONGLONG              ullOffset,
                [in] VDS_FILE_SYSTEM_TYPE   type,
                [in, string] WCHAR          *pwszLabel,
                [in] DWORD                  dwUnitAllocationSize,
                [in] long                   bForce,
                [in] long                   bQuickFormat,
                [in] long                   bEnableCompression,
                [out] IVdsAsync             **ppAsync
            );

        HRESULT Clean(
                [in] long       bForce,
                [in] long       bForceOEM,
                [in] long       bFullClean,
                [out] IVdsAsync **ppAsync
            );
    }

    [
        object,
        uuid(9723f420-9355-42de-ab66-e31bb15beeac),
        pointer_default(unique)
    ]
    interface IVdsAdvancedDisk2 : IUnknown
    {
        [helpstring("method ChangePartitionType")]
        HRESULT ChangePartitionType(
                [in] ULONGLONG                          ullOffset,
                [in] long                               bForce,
                [in] CHANGE_PARTITION_TYPE_PARAMETERS   * para
            );
    }

    [
        object,
        uuid(9882f547-cfc3-420b-9750-00dfbec50662),
        pointer_default(unique)
    ]
    interface IVdsCreatePartitionEx : IUnknown
    {
        [helpstring("method CreatePartitionEx")]
        HRESULT CreatePartitionEx(
                [in] ULONGLONG                  ullOffset,
                [in] ULONGLONG                  ullSize,
                [in] unsigned long              ulAlign,
                [in] CREATE_PARTITION_PARAMETERS  *para,
                [out] IVdsAsync                 **ppAsync
            );
    }

    [
        object,
```

```
    uuid(538684e0-ba3d-4bc0-aca9-164aff85c2a9),
    pointer_default(unique)
]
interface IVdsDiskPartitionMF : IUnknown
{
    [helpstring("method GetPartitionFileSystemProperties")]
    HRESULT GetPartitionFileSystemProperties(
                [in] ULONGLONG              ullOffset,
                [out] VDS_FILE_SYSTEM_PROP  *pFileSystemProp
            );

    [helpstring("method GetPartitionFileSystemTypeName")]
    HRESULT GetPartitionFileSystemTypeName(
                [in] ULONGLONG      ullOffset,
                [out, string] WCHAR **ppwszFileSystemTypeName
            );

    [helpstring("method QueryPartitionFileSystemFormatSupport")]
    HRESULT QueryPartitionFileSystemFormatSupport(
                [in] ULONGLONG
     ullOffset,
                [out, size_is(,*plNumberOfFileSystems)]
                      VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP
     **ppFileSystemSupportProps,
                [out] long
     *plNumberOfFileSystems
            );

    [helpstring("method FormatPartitionEx")]
    HRESULT FormatPartitionEx(
                [in] ULONGLONG              ullOffset,
                [in, unique, string] WCHAR
     *pwszFileSystemTypeName,
                [in] unsigned short         usFileSystemRevision,
                [in] unsigned long
     ulDesiredUnitAllocationSize,
                [in, unique, string] WCHAR  *pwszLabel,
                [in] long                   bForce,
                [in] long                   bQuickFormat,
                [in] long                   bEnableCompression,
                [out] IVdsAsync             **ppAsync
            );
}

[
    object,
    uuid(0316560b-5db4-4ed9-bbb5-213436ddc0d9),
    pointer_default(unique)
]
interface IVdsRemovable : IUnknown
{
    [helpstring("method QueryMedia")]
    HRESULT QueryMedia();

    [helpstring("method Eject")]
    HRESULT Eject();
}
```

```
[
    object,
    uuid(88306bb2-e71f-478c-86a2-79da200a0f11),
    pointer_default(unique)
]
interface IVdsVolume : IUnknown
{
    [helpstring("method GetProperties")]
    HRESULT GetProperties(
                [out] VDS_VOLUME_PROP   *pVolumeProperties
            );

    [helpstring("method GetPack")]
    HRESULT GetPack(
                [out] IVdsPack  **ppPack
            );

    [helpstring("method QueryPlexes")]
    HRESULT QueryPlexes(
                [out] IEnumVdsObject    **ppEnum
            );

    [helpstring("method Extend")]
    HRESULT Extend(
                [in, unique, size_is(lNumberOfDisks)]
                     VDS_INPUT_DISK *pInputDiskArray,
                [in] long           lNumberOfDisks,
                [out] IVdsAsync     **ppAsync
            );

    [helpstring("method Shrink")]
    HRESULT Shrink(
                [in] ULONGLONG  ullNumberOfBytesToRemove,
                [out] IVdsAsync **ppAsync
            );

    [helpstring("method AddPlex")]
    HRESULT AddPlex(
                [in] VDS_OBJECT_ID  VolumeId,
                [out] IVdsAsync     **ppAsync
            );

    [helpstring("BreakPlex")]
    HRESULT BreakPlex(
                [in] VDS_OBJECT_ID  plexId,
                [out] IVdsAsync     **ppAsync
            );

    [helpstring("RemovePlex")]
    HRESULT RemovePlex(
                [in] VDS_OBJECT_ID  plexId,
                [out] IVdsAsync     **ppAsync
            );

    [helpstring("method Delete")]
    HRESULT Delete(
                [in] long   bForce
            );
```

```
    [helpstring("method SetFlags")]
    HRESULT SetFlags(
                [in] unsigned long  ulFlags,
                [in] long           bRevertOnClose
            );

    [helpstring("method ClearFlags")]
    HRESULT ClearFlags(
                [in] unsigned long  ulFlags
            );
}

[
    object,
    uuid(ee2d5ded-6236-4169-931d-b9778ce03dc6),
    pointer_default(unique)
]
interface IVdsVolumeMF : IUnknown
{
    [helpstring("method queryFileSystemProperties")]
    HRESULT GetFileSystemProperties(
                [out] VDS_FILE_SYSTEM_PROP  *pFileSystemProp
            );

    [helpstring("method Format")]
    HRESULT Format(
                [in] VDS_FILE_SYSTEM_TYPE   type,
                [in, string] WCHAR          *pwszLabel,
                [in] DWORD                  dwUnitAllocationSize,
                [in] long                   bForce,
                [in] long                   bQuickFormat,
                [in] long                   bEnableCompression,
                [out] IVdsAsync             **ppAsync
            );

    [helpstring("method AddAccessPath")]
    HRESULT AddAccessPath(
                [in, max_is(MAX_PATH - 1), string] WCHAR
     *pwszPath
            );

    [helpstring("method QueryAccessPaths")]
    HRESULT QueryAccessPaths(
                [out, string, size_is(, *plNumberOfAccessPaths)]
                      WCHAR ***pppwszPathArray,
                [out] long  *plNumberOfAccessPaths
            );

    [helpstring("method QueryReparsePoints")]
    HRESULT QueryReparsePoints(
                [out, size_is(, *plNumberOfReparsePointProps)]
                    VDS_REPARSE_POINT_PROP **ppReparsePointProps,
                [out] long
     *plNumberOfReparsePointProps
            );

    [helpstring("method DeleteAccessPath")]
```

```
    HRESULT DeleteAccessPath(
        [in, max_is(MAX_PATH - 1), string] WCHAR
     *pwszPath,
        [in] long           bForce
            );

    [helpstring("method Mount")]
    HRESULT Mount();

    [helpstring("method Dismount")]
    HRESULT Dismount(
                [in] long   bForce,
                [in] long   bPermanent
            );

    [helpstring("method SetFileSystemFlags")]
    HRESULT SetFileSystemFlags(
                [in] unsigned long  ulFlags
            );

    [helpstring("method ClearFileSystemFlags")]
    HRESULT ClearFileSystemFlags(
                [in] unsigned long  ulFlags
            );
}

[
    object,
    uuid(4dbcee9a-6343-4651-b85f-5e75d74d983c),
    pointer_default(unique)
]
interface IVdsVolumeMF2 : IUnknown
{
    [helpstring("method GetFileSystemTypeName")]
    HRESULT GetFileSystemTypeName(
                [out, string] WCHAR **ppwszFileSystemTypeName
            );

    [helpstring("method QueryFileSystemFormatSupport")]
    HRESULT QueryFileSystemFormatSupport(
        [out, size_is(,*plNumberOfFileSystems)]
                        VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP
     **ppFileSystemSupportProps,
        [out] long    *plNumberOfFileSystems
            );

    [helpstring("method FormatEx")]
    HRESULT FormatEx(
             [in, unique, string] WCHAR  *pwszFileSystemTypeName,
             [in] unsigned short         usFileSystemRevision,
             [in] unsigned long
     ulDesiredUnitAllocationSize,
             [in, unique, string] WCHAR  *pwszLabel,
             [in] long                   bForce,
             [in] long                   bQuickFormat,
             [in] long                   bEnableCompression,
             [out] IVdsAsync             **ppAsync
            );
```

```
}

[
    object,
    uuid(d68168c9-82a2-4f85-b6e9-74707c49a58f),
    pointer_default(unique)
]
interface IVdsVolumeShrink : IUnknown
{
    [helpstring("method QueryMaxReclaimableBytes")]
    HRESULT QueryMaxReclaimableBytes(
                [out] ULONGLONG *pullMaxNumberOfReclaimableBytes
    );

    [helpstring("method Shrink")]
    HRESULT Shrink(
        [in] ULONGLONG  ullDesiredNumberOfReclaimableBytes,
        [in] ULONGLONG  ullMinNumberOfReclaimableBytes,
        [out] IVdsAsync **ppAsync
    );
}

[
    object,
    uuid(1BE2275A-B315-4f70-9E44-879B3A2A53F2),
    pointer_default(unique)
]

interface IVdsVolumeOnline : IUnknown
{
    [helpstring("method Online")]
    HRESULT Online();
}

[
    object,
    uuid(4daa0135-e1d1-40f1-aaa5-3cc1e53221c3),
    pointer_default(unique)
]
interface IVdsVolumePlex : IUnknown
{
    [helpstring("method GetProperties")]
    HRESULT GetProperties(
                [out] VDS_VOLUME_PLEX_PROP  *pPlexProperties
            );

    [helpstring("method GetVolume")]
    HRESULT GetVolume(
                [out] IVdsVolume    **ppVolume
            );

    [helpstring("method QueryExtents")]
    HRESULT QueryExtents(
                [out, size_is(, *plNumberOfExtents)]
                      VDS_DISK_EXTENT   **ppExtentArray,
                [out] long              *plNumberOfExtents
            );
```

```
    [helpstring("method Repair")]
    HRESULT Repair(
                [in, size_is(lNumberOfDisks)]
                     VDS_INPUT_DISK *pInputDiskArray,
                [in] long           lNumberOfDisks,
                [out] IVdsAsync     **ppAsync
            );
}
```

7 Appendix B: Windows Behavior

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

- Windows Server 2003
- Windows Server 2003 SP1
- Windows Server 2003 R2
- Windows Vista
- Windows Server 2008

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.3: For a server, the VDS Protocol is implemented by the Windows VDS on Windows Server 2003, Windows Server 2003 R2, Windows Vista, and Windows Server 2008. For the client, the VDS Protocol is implemented by a number of components, including the Windows logical disk manager user interface and the DISKPART.EXE command line tool. This protocol is supported on Windows Server 2003, Windows Server 2003 R2, Windows Vista, and Windows Server 2008. Windows XP and Windows 2000 do not support VDS, but the VDS Protocol on those operating systems provides a subset of the functionality that VDS provides.

<2> Section 1.6: Windows 2000, Windows XP, and Windows Server 2003 implement DMRP, as specified in [MS-DMRP]. Windows Server 2003 and later operating systems implement the VDS Protocol. The VDS Protocol is the preferred protocol to use with Windows Server 2003 and later. The interfaces that are associated with DMRP, as specified in [MS-DMRP], are not available on Windows Vista and later releases.

<3> Section 1.7: interfaces common to all storage object management.

The following table lists, by operating system version, the interfaces common to all storage management.

	Supported operating system version			
Interface	**Windows Server 2003**	**Windows Server 2003 R2**	**Windows Vista**	**Windows Server 2008**
IEnumVdsObject	X	X	X	X
IVdsAdviseSink	X	X	X	
IVdsAsync	X	X	X	X
IVdsService	X	X	X	X
IVdsServiceInitialization	X		X	X

Interfaces that are used when managing disks and volumes.

The following table lists, by operating system version, the interfaces used when managing disks and volumes.

	Supported OS versions			
Interface	**Windows Server 2003**	**Windows Server 2003 R2**	**Windows Vista**	**Windows Server 2008**
IVdsProvider	X	X		
IVdsSwProvider	X	X		
IVdsPack	X	X	X	X
IVdsDisk	X	X		X
IVdsAdvancedDisk	X	X	X	X
IVdsCreatePartitionEx	X	X	X	X
IVdsRemovable	X	X	X	X
IVdsVolume	X	X	X	X
IVdsVolumeMF	X	X	X	X
IVdsVolumePlex	X	X	X	
IVdsServiceUninstallDisk		X	X	X
IVdsPack2			X	X
IVdsDisk2			X	X
IVdsAdvancedDisk2			X	X
IVdsDiskPartitionMF			X	X
IVdsVolumeMF2			X	X
IVdsVolumeShrink			X	X
IVdsVolumeOnline			X	X

The following list provides a brief overview of the functionality available for managing disks and volumes in the initial release of the VDS Protocol in Windows Server 2003:

IVdsAdvancedDisk2 extends **IVdsAdvancedDisk** with new functionality that is related to changing a partition type.

IVdsServiceUninstallDisk contains new functionality that is related to uninstalling disks and the volumes that are contained on those disks.

IVdsPack2 extends **IVdsPack** with new functionality that is related to creating aligned volumes.

IVdsDisk2 extends **IVdsDisk** with new functionality that is related to bringing disks online and offline in clustered and other scenarios.

IVdsVolumeMF2 extends **IVdsVolumeMF** with new functionality that is related to file systems.

IVdsDiskPartitionMF contains new functionality that is related to file systems.

IVdsVolumeShrink contains new functionality that is related to shrinking volumes.

IVdsVolumeOnline contains new functionality that is related to bringing volumes online.

Interfaces that are used when querying HBA and iSCSI initiator information.

The following table lists, by operating system version, the interfaces that are used when querying HBA and iSCSI initiator information.

	Supported OS versions			
Interface	**Windows Server 2003**	**Windows Server 2003 R2**	**Windows Vista**	**Windows Server 2008**
IVdsServiceHba		X	X	X
IVdsServiceIscsi		X	X	X
IVdsHbaPort		X	X	X
IVdsIscsiInitiatorAdapter		X	X	X
IVdsIscsiInitiatorPortal		X	X	X

<4> Section 2.1: Windowsconfigures the underlying RPC transport by using the following flags. For more information on the meaning of these flags, see [C706] and [MS-RPCE].

- RPC_C_AUTHN_LEVEL_PKT_PRIVACY
- RPC_C_IMP_LEVEL_IDENTIFY
- EOAC_SECURE_REFS | EOAC_NO_CUSTOM_MARSHAL

<5> Section 2.1: The authorization constraints in Windows do not vary by operating system release. All interfaces that are described in this document require an access level that corresponds to any of the following Windows security groups:

- Administrators
- Backup Operators
- SYSTEM

<6> Section 2.2: Unless otherwise stated, all data types and messages for the VDS Protocol are supported in Windows Server 2003 and later.

<7> Section 2.2.1.2.11: This enumeration type is

used only in Windows Vista.

<8> Section 2.2.1.3.11: Windows recognizes the following partition types on MBR disks and treats all others as OEM partitions (which do not automatically get assigned drive letters except in **Windows Preinstallation Environment (Windows PE)**):

Value	Meaning
0x00	An unused entry.
0x01	Partition with 12-bit FAT entries.
0x04	Partition with 16-bit FAT entries.
0x05	Extended partition entry.
0x06	MS-DOS version 4 huge partition.
0x07	Installable file system (IFS) partition.
0x0B	FAT32 partition.
0x0C	FAT32 partition using extended INT13 services.
0x0E	16-bit FAT partition using extended INT13 services.
0x0F	Extended partition using extended INT13 services.
0x42	Logical disk manager (LDM) data partition.

<9> Section 2.2.1.3.12: Windows recognizes the following partition types on GPTdisks:

Value	Meaning
{C12A7328-F81F-11D2-BA4B-00A0C93EC93B}	EFI system partition.
{E3C9E316-0B5C-4DB8-817D-F92DF00215AE}	MSR space partition; used to reserve space for subsequent use by operating system software.
{EBD0A0A2-B9E5-4433-87C0-68B6B72699C7}	A basic data partition.

Value	Meaning
{5808C8AA-7E8F-42E0-85D2-E1E90434CFB3}	LDM metadata partition.
{AF9B60A0-1431-4F62-BC68-3311714A69AD}	LDM data partition.
{DE94BBA4-06D1-4D40-A16A-BFD50179D6AC}	Microsoft recovery partition.

<10> Section 2.2.1.3.12: The partition is recognized as an OEM partition and will not be converted to dynamic if the disk is converted to dynamic. The partition will not get a drive letter except in the Windows PE.

<11> Section 2.2.1.3.18: This structure is used only in Windows Vista.

<12> Section 2.2.1.3.22: Only the basic data partition type is allowed.

<13> Section 2.2.1.3.22: The partition is recognized as an OEM partition and is not converted to dynamic if the disk is converted to dynamic. The partition does not get a drive letter except in Windows PE.

<14> Section 2.2.2.1.2.3: VDS_QUERY_HARDWARE_PROVIDERS returns no additional providers when it is used with **IVdsService::QueryProviders(Opnum 6)** on a Windows installation that has no additional software installed. Third-party providers must be installed to get hardware providers.

<15> Section 2.2.2.1.3.1: Windows Server 2003 has **pwszVersion** set to the string "1.0". Windows Server 2003 R2 has **pwszVersion** set to the string "1.1". Windows Vista and Windows Server 2008 have **pwszVersion** set to the string "2.0".

<16> Section 2.2.2.2.1.1: This structure is used only in Windows Server 2003 R2 and later.

<17> Section 2.2.2.3.1.2: This enumeration type is used only in Windows Server 2003 R2 and later.

<18> Section 2.2.2.3.1.3: This enumeration type is used only in Windows Server 2003 R2 and later.

<19> Section 2.2.2.3.2.1: This structure is only used in Windows Server 2003 R2 and Windows Server 2008.

<20> Section 2.2.2.3.2.2: This structure is used only in Windows Server 2003 R2 and later.

<21> Section 2.2.2.4.1.1: This structure is used only in Windows Server 2003 R2 and later. Attempting to call these methods with opnums 5 or 6 MAY result in Network Data Representation (NDR) raising an RPC_X_BAD_STUB_DATA exception. For more information, see [MS-DCOM].

<22> Section 2.2.2.5.1.1: This enumeration type is used only in Windows Server 2003 R2 and later.

<23> Section 2.2.2.5.2.1: This structure is used only in Windows Server 2003 R2 and later.

<24> Section 2.2.2.5.2.2: This structure is used only in Windows Server 2003 R2 and later.

<25> Section 2.2.2.8.2.1: This string is used to load the property page information for a disk.

<26> Section 2.2.2.9.1.1: The partition is recognized as an OEM partition and is not converted to dynamic if the disk is converted to dynamic. The partition does not get a drive letter except in Windows PE.

<27> Section 2.2.2.10.1.1: This structure is used only in Windows Vista.

<28> Section 2.2.2.10.1.1: Only the basic data partition type is allowed.

<29> Section 2.2.2.11.1.2: This flag is not valid for **basic** and dynamic volumes and is only supported by some third-party volume managers.

<30> Section 3.1.5.1: Attempting to call a method with opnum 7 may result in NDR raising a RPC_X_BAD_STUB_DATA exception. For more information, see [MS-DCOM].

<31> Section 3.1.5.1: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
07	Only used locally by Windows, never used remotely.

<32> Section 3.1.5.3: This interface is only supported in Windows Server 2003 R2 and later. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned.

<33> Section 3.1.5.4: This interface is only supported in Windows Server 2003 R2 and later. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned.

<34> Section 3.1.5.5: This interface is only supported in Windows Server 2003 R2 and later. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned. Attempting to call methods with opnums 5, 6, 7, or 9 may result in NDR raising a RPC_X_BAD_STUB_DATA exception. For more information, see [MS-DCOM].

<35> Section 3.1.5.5: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
05	Only used locally by Windows, never used remotely.
06	Only used locally by Windows, never used remotely.
07	Only used locally by Windows, never used remotely.
09	Only used locally by Windows, never used remotely.

<36> Section 3.1.6.1: This interface is only supported in Windows Server 2003 R2 and later. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned.

<37> Section 3.1.7.1: This interface is only supported in Windows Server 2003 R2 and later. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned. Attempting to call these methods with opnums 5 or 6 may result in NDR raising an RPC_X_BAD_STUB_DATA exception. For more information, see [MS-DCOM].

<38> Section 3.1.7.1: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
05	Only used locally by Windows, never used remotely.
06	Only used locally by Windows, never used remotely.

<39> Section 3.1.8.1: This interface is only supported in Windows Server 2003 R2 and later. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned. Attempting to call methods with opnums 5, 6, or 7 may result in NDR raising an RPC_X_BAD_STUB_DATA exception. For more information, see [MS-DCOM].

<40> Section 3.1.8.1: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
05	Only used locally by Windows, never used remotely.
06	Only used locally by Windows, never used remotely.
07	Only used locally by Windows, never used remotely.

<41> Section 3.1.11.1: Attempting to call a method with opnum 10 may result in NDR raising an RPC_X_BAD_STUB_DATA exception. For more information, see [MS-DCOM].

<42> Section 3.1.11.1: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
10	Only used locally by Windows, never used remotely.

<43> Section 3.1.11.2: This interface is supported only in Windows Vista. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned.

<44> Section 3.1.12.1: Attempting to call methods with opnums 8 or 9 may result in NDR raising an RPC_X_BAD_STUB_DATA exception. For more information, see [MS-DCOM].

<45> Section 3.1.12.2: This interface is only supported in Windows Vista. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned.

<46> Section 3.1.12.4: This interface is only supported in Windows Vista. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned.

<47> Section 3.1.12.6: This interface is only supported in Windows Vista. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned.

<48> Section 3.1.13.3: This interface is only supported in Windows Vista. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned.

<49> Section 3.1.13.4: This interface is only supported in Windows Vista. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned.

<50> Section 3.1.13.5: This interface is only supported in Windows Vista. If a client attempts to get the interface on a previous version of Windows by calling IUnknown::QueryInterface, E_NOINTERFACE (0x80004002) is returned.

<51> Section 3.3.1.2: In Windows, only partitions on hard disks that have the following partition types are considered volumes: 0x01, 0x04, 0x06, 0x07, 0x0B, 0x0C, and 0x0E.

<52> Section 3.3.5.2: Windows servers enforce authorization checks. For information on the authorization requirements for the various methods, see section 2.1. On Windows, the client must be a member of the Administrator or backup operators groups, or be the local_system account.

<53> Section 3.3.5.2.6.1: This method is supported only in Windows Server 2003 R2 and later.

<54> Section 3.3.5.2.6.2: This method is supported only in Windows Server 2003 R2 and later.

<55> Section 3.3.5.2.7.1: This method is only supported in Windows Server 2003 R2 and later.

<56> Section 3.3.5.2.8.1: This method is only supported in Windows Server 2003 R2 and later.

<57> Section 3.3.5.2.8.2: This method is only supported in Windows Server 2003 R2 and later.

<58> Section 3.3.5.2.8.3: This method is supported only in Windows Server 2003 R2 and later. The Windows implementation of this method always returns VDS_E_TARGET_SPECIFIC_NOT_SUPPORTED (0x80042706) if a target ID is specified.

<59> Section 3.3.5.2.9.1: This method is supported only in Windows Server 2003 R2 and later.

<60> Section 3.3.5.2.9.2: This method is only supported in Windows Server 2003 R2 and later.

<61> Section 3.3.5.2.10.1: This method is only supported in Windows Server 2003 R2 and later.

<62> Section 3.3.5.2.10.2: This method is supported only in Windows Server 2003 R2 and later.

<63> Section 3.3.5.2.11.1: This method is only supported in Windows Server 2003 R2 and later.

<64> Section 3.3.5.2.11.2: This method is supported only in Windows Server 2003 R2 and later.

<65> Section 3.3.5.2.16.5: Windows supports at most 32 disks in a volume. Windows servers fail requests that specify more than 32 disks, and Windows clients never submit such requests.

<66> Section 3.3.5.2.16.5: The Windows implementation requires that the stripe size MUST be 65536 if the type is VDS_VT_STRIPE or VDS_VT_PARITY. Other volume types are not striped, and therefore the stripe size MUST be 0 for other volume types.

<67> Section 3.3.5.2.16.6: If you add a GPT disk to a basic pack, the operation automatically creates an MSR partition on the disk, except when the server is running in the Windows PE because an administrator may want to create an EFI system partition on the disk. The EFI system partition, if present, must be the first partition on the disk. If a disk is added to a dynamic pack, the operation does not create an MSR partition.

<68> Section 3.3.5.2.17.1: This array's size MUST be 32 objects or less, because Windows imposes a limit of 32 disks that can be used with a single volume.

<69> Section 3.3.5.2.17.1: The stripe size MUST be 65,536 if type is VDS_VT_STRIPE or VDS_VT_PARITY; otherwise, stripe size MUST be 0.

<70> Section 3.3.5.2.17.1: In Windows, if zero is specified, the server determines the alignment value that is specified in one of the following registry keys under HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\vds\Alignment, depending on the size of the disk on which the volume is created. The following default values appear after the operating system is installed and may be overridden by an administrator.

Value	Meaning
"Disk Size 4GB" 65536	Registry Key: LessThan4GB
"Disk Size 4 - 8GB" 1048576	Registry Key: Between4_8GB
"Disk Size 8 - 32GB" 1048576	Registry Key: Between8_32GB
"Disk Size > 32GB" 1048576	Registry Key: GreaterThan32GB

<71> Section 3.3.5.2.19.1: This method is only supported in Windows Vista.

<72> Section 3.3.5.2.20.4: OEM partitions, EFI system partitions, and MSR partitions are considered protected and cannot be deleted unless **bForceProtected** is specified.

<73> Section 3.3.5.2.20.9: This parameter is ignored if the file system is not an NTFS file system.

<74> Section 3.3.5.2.20.10: On GPT disks, Windows recognizes any partition with the GPT_ATTRIBUTE_PLATFORM_REQUIRED flag set as an OEM partition. On MBR disks, Windows recognizes the following partition types as OEM partitions:

Value	Meaning
0x12	EISA partition.
0x27	Microsoft recovery partition (only recognized in Windows Vista).
0x84	Hibernation partition for laptops.
0xA0	Diagnostic partition on some HP notebooks.
0xDE	Dell partition.
0xFE	IBM IML partition.

<75> Section 3.3.5.2.22.1: For MBR-formatted disks, partition size is limited to 2^32 sectors. For example, for a sector size of 512 bytes, there is a 2-terabyte limit for partitions.

<76> Section 3.3.5.2.23.1: This method is supported only in Windows Vista.

<77> Section 3.3.5.2.23.2: This method is only supported in Windows Vista.

<78> Section 3.3.5.2.23.3: This method is supported only in Windows Vista.

<79> Section 3.3.5.2.23.4: This method is supported only in Windows Vista.

<80> Section 3.3.5.2.23.4: This parameter is ignored if the file system is not an NTFS file system.

<81> Section 3.3.5.2.25.4: No more than 32 disks MAY be used with a volume; therefore, this value MUST be no more than 31.

<82> Section 3.3.5.2.25.5: Only an NTFS file system or raw volumes support this operation.

<83> Section 3.3.5.2.25.5: Shrink will only work on volumes with an NTFS file system or RAW file systems; otherwise, returns VDS_E_CANNOT_SHRINK (HRESULT of 0x8004251E).

<84> Section 3.3.5.2.25.9: Windows Server 2003: The crash dump and hibernate volumes MUST remain on the **boot partition**.

Windows Server 2003: After the volume is deleted, VDS tries to delete the volume mount points. If deleting the mount points fails, Delete will return VDS_S_ACCESS_PATH_NOT_DELETED, even though the volume was successfully deleted.

<85> Section 3.3.5.2.25.10: In Windows Vista and Windows Server 2008, if flags VDS_VF_READ_ONLY and/or VDS_VF_HIDDEN are being set and **bRevertOnClose** is not set, the server dismounts the volume so that the new flags take effect when the volume is mounted.

<86> Section 3.3.5.2.25.10: If **bRevertOnClose** is set, the server does not dismount the volume, and the new flags do not take effect unless the machine is rebooted, or unless the volume is dismounted and mounted by the client before releasing the last reference to the volume.

<87> Section 3.3.5.2.26.2: This parameter is ignored if the file system is not an NTFS file system.

<88> Section 3.3.5.2.26.3: An access path can apply to a drive letter or an empty folder on an NTFS file system.

<89> Section 3.3.5.2.27.1: This method is supported only in Windows Vista.

<90> Section 3.3.5.2.27.2: This method is only supported in Windows Vista.

<91> Section 3.3.5.2.27.3: This parameter is ignored if the file system is not an NTFS file system.

<92> Section 3.3.5.2.28.1: This method is supported only in Windows Vista.

<93> Section 3.3.5.2.28.2: This method is supported only in Windows Vista.

<94> Section 3.3.5.2.28.2: The **Shrink** method works only on volumes that have NTFS or RAW file systems; otherwise, **Shrink** returns VDS_E_CANNOT_SHRINK (HRESULT of 0x8004251E).

<95> Section 3.3.5.2.29.1: This method is supported only in Windows Vista.

<96> Section 3.3.5.2.30.4: Only one new disk can be passed to this method at a time.

<97> Section 3.3.5.2.30.4: Only one new disk can be passed to this method at a time.

<98> Section 3.3.7: Windows VDS Protocol servers use the Plug and Play subsystem to register with the operating system to receive notifications of changes to the storage device.

8 Index

A

Abstract data model
  client
  server
Access paths
AddAccessPath method
AddDisk method
AddPlex method
Advise method
Applicability
AssignDriveLetter method
Asynchronous operation object interfaces
Asynchronous tasks
Asynchronous tasks - performing - example

B

BreakPlex method

C

Callback object interface
Callback objects
Cancel method
Capability negotiation
CHANGE_ATTRIBUTES_PARAMETERS structure
CHANGE_PARTITION_TYPE_PARAMETERS structure
ChangeAttributes method
ChangePartitionType method
Clean method
CleanupObsoleteMountPoints method
ClearFileSystemFlags method
ClearFlags method (section 3.3.5.2.4.16, section 3.3.5.2.18.7, section 3.3.5.2.25.11)
Client
  abstract data model
  initialization
  local events
  message processing
  notifications examples
  overview
  sequencing rules
  timer events
  timers
Clone method
Common data types
ConvertStyle method
CREATE_PARTITION_PARAMETERS structure
CreatePack method
CreatePartition method
CreatePartitionEx method
CreateVolume method
CreateVolume2 method

D

Data model - abstract
  client
  server
Data types
  common
  interface-specific
  IVdsAdvancedDisk
  IVdsAdvancedDisk2
  IVdsDisk
  IVdsHbaPort
  IVdsIscsiInitiatorAdapter
  IVdsIscsiInitiatorPortal
  IVdsPack
  IVdsProvider
  IVdsService (section 2.2.2.1, section 2.2.2.1.1)
  IVdsServiceIscsi
  IVdsVolume
  IVdsVolumeMF (section 2.2.2.12, section 2.2.2.12.1)
  IVdsVolumePlex
  overview
Delete method
DeleteAccessPath method
DeleteDriveLetter method
DeletePartition method
Disk
  arrival
  removal
Disk object interfaces
Disk pack
  arrival
  removal
Disks
Dismount method
Drive letter
  assignment
  removal
Drive letters
DWORD

E

Eject method
Enumeration - objects
Enumeration object interface
Enumerations
  common data types
  IVdsDisk
  IVdsHbaPort
  IVdsIscsiInitiatorPortal
  IVdsPack
  IVdsProvider
  IVdsService
  IVdsVolume
  IVdsVolumePlex
  querying example
Examples
  overview
  performing asynchronous tasks
  querying enumerations of VDS objects
  retrieving properties and IDs
  VDS client notifications

VDS sessions
Extend method

F

Fields - vendor-extensible
File system modification
File systems
Format method
FormatEx method
FormatPartition method
FormatPartitionEx method
Full IDL

G

GetDiskIdFromLunInfo method
GetDriveLetter method
GetFileSystemProperties method
GetFileSystemTypeName method
GetIdentificationData method
GetImportTarget method
GetInitiatorAdapter method
GetInitiatorName method
GetObject method
GetPack method (section 3.3.5.2.18.2, section 3.3.5.2.25.2)
GetPartitionFileSystemProperties method
GetPartitionFileSystemTypeName method
GetPartitionProperties method
GetProperties method (section 3.3.5.2.4.3, section 3.3.5.2.9.1, section 3.3.5.2.10.1, section 3.3.5.2.11.1, section 3.3.5.2.12.1, section 3.3.5.2.16.1, section 3.3.5.2.18.1, section 3.3.5.2.25.1, section 3.3.5.2.30.1)
GetProvider method
GetVolume method
Glossary

H

HBA port object interfaces
Hierarchy overview
Higher-layer triggered events - server

I

ID retrieval example
IDL
IEnumVdsObject interface
IEnumVdsObject methods
Implementers - security considerations
Informative references
Initialization
client
server
Initialize method
Initiator object interfaces
adapter
portal
Interfaces
asynchronous operation object
callback object
disk object
enumeration object
HBA port object
IEnumVdsObject
initiator adapter object
initiator portal object
IVdsAdvancedDisk
IVdsAdvancedDisk2
IVdsAdviseSink
IVdsAsync
IVdsCreatePartitionEx
IVdsDisk
IVdsDiskPartitionMF
IVdsDisks
IVdsHbaPort
IVdsIscsiInitiatorAdapter
IVdsIscsiInitiatorPortal
IVdsPack
IVdsPack2
IVdsProvider
IVdsRemovable
IVdsService
IVdsServiceHba
IVdsServiceInitialization
IVdsServiceIscsi
IVdsServiceUninstallDisk
IVdsSwProvider
IVdsVolume
IVdsVolumeMF
IVdsVolumeMF2
IVdsVolumeOnline
IVdsVolumePlex
IVdsVolumeShrink
overview
pack object
service object
software provider object
volume object
volume plex object
Interfaces - data types
Introduction
IsServiceReady method
IVdsAdvancedDisk
data types
interface
methods
structures
IVdsAdvancedDisk2
data types
interface
methods
structures
IVdsAdviseSink
interface
methods
IVdsAsync
interface
methods
IVdsCreatePartitionEx
interface

methods
IVdsDisk
data types
enumerations
interface
methods
structures
IVdsDisk2
interface
methods
IVdsDiskPartitionMF
interface
methods
IVdsHbaPort
data types
enumerations
interface
methods
structures
IVdsIscsiInitiatorAdapter
data types
interface
methods
structures
IVdsIscsiInitiatorPortal
data types
enumerations
interface
methods
structures
IVdsPack
data types
enumerations
interface
methods
structures
IVdsPack2
interface
methods
IVdsProvider
data types
enumerations
interface
methods
structures
IVdsRemovable
interface
methods
IVdsService
data types
enumerations
interface
methods
structures
IVdsService data types - overview
IVdsServiceHba
interface
methods
IVdsServiceInitialization
interface
methods
IVdsServiceIscsi
data types
interface
methods
structures
IVdsServiceUninstallDisk
interface
methods
IVdsSwProvider
interface
methods
IVdsVolume
data types
enumerations
interface
methods
structures
IVdsVolumeMF
data types (section 2.2.2.12, section 2.2.2.12.1)
interface
methods
structures
IVdsVolumeMF2
interface
methods
IVdsVolumeOnline
interface
methods
IVdsVolumePlex
data types
enumerations
interface
methods
structures
IVdsVolumeShrink
interface
methods

L

LoadService method
Local events
client
server

M

MAX_FS_NAME_SIZE
MAX_PATH
Media
arrival
removal
Message processing
client
server
Messages
common data types (section 2.2.1, section 2.2.1.1)
enumerations
interface-specific data types
IVdsAdvancedDisk data types
IVdsAdvancedDisk structures
IVdsAdvancedDisk2 data types

IVdsAdvancedDisk2 structures
IVdsDisk data types
IVdsDisk enumerations
IVdsDisk structures
IVdsHbaPort data types
IVdsHbaPort enumerations
IVdsHbaPort structures
IVdsIscsiInitiatorAdapter data types
IVdsIscsiInitiatorAdapter structures
IVdsIscsiInitiatorPortal data types
IVdsIscsiInitiatorPortal enumerations
IVdsIscsiInitiatorPortal structures
IVdsPack data types
IVdsPack enumerations
IVdsPack structures
IVdsProvider data types
IVdsProvider enumerations
IVdsProvider structures
IVdsService data types (section 2.2.2.1, section 2.2.2.1.1)
IVdsService enumerations
IVdsService structures
IVdsServiceIscsi data types
IVdsServiceIscsi structures
IVdsVolume data types
IVdsVolume enumerations
IVdsVolume structures
IVdsVolumeMF data types (section 2.2.2.12, section 2.2.2.12.1)
IVdsVolumeMF structures
IVdsVolumePlex data types
IVdsVolumePlex enumerations
IVdsVolumePlex structures
overview
structures
syntax
transport
Methods
IEnumVdsObject
invocation
IVdsAdvancedDisk
IVdsAdvancedDisk2
IVdsAdviseSink
IVdsAsync
IVdsCreatePartitionEx
IVdsDisk
IVdsDisk2
IVdsDiskPartitionMF
IVdsHbaPort
IVdsIscsiInitiatorAdapter
IVdsIscsiInitiatorPortal
IVdsPack
IVdsPack2
IVdsProvider
IVdsRemovable
IVdsService
IVdsServiceHba
IVdsServiceInitialization
IVdsServiceIscsi
IVdsServiceUninstallDisk
IVdsSwProvider
IVdsVolume
IVdsVolumeMF
IVdsVolumeMF2
IVdsVolumeOnline
IVdsVolumePlex
IVdsVolumeShrink
sequencing requirements
MigrateDisks method
Mount method
Mount print change

N

Next method
Normative references
Notification callback objects (section 3.2.1.1, section 3.3.1.4, section 3.3.3.2)

O

Object enumeration
Online method
OnNotify method
Overview

P

Pack object interfaces
Packs
Parameters - security
Paths - access
Preconditions
Prerequisites
Processing notifications from server to client
Processing server replies to method calls
Properties retrieval example
Providers
PVDS_DISK_EXTENT
PVDS_DISK_PROP
PVDS_DRIVE_LETTER_PROP
PVDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP
PVDS_FILE_SYSTEM_PROP
PVDS_FILE_SYSTEM_TYPE_PROP
PVDS_PACK_PROP
PVDS_REPARSE_POINT_PROP
PVDS_VOLUME_PLEX_PROP
PVDS_VOLUME_PROP

Q

QueryAccessPaths method
QueryDisks method
QueryDriveLetters method
QueryExtents method (section 3.3.5.2.18.4, section 3.3.5.2.30.3)
QueryFileSystemFormatSupport method
QueryFileSystemTypes method
QueryHbaPorts method
QueryInitiatorAdapters method
QueryInitiatorPortals method
QueryMaxReclaimableBytes method
QueryMedia method

QueryPacks method
QueryPartitionFileSystemFormatSupport method
QueryPartitions method
QueryPlexes method
QueryProviders method
QueryReparsePoints method
QueryStatus method
QuerySubSystems method
QueryUnallocatedDisks method
QueryVolumes method

R

Reboot method
Recover method
Reenumerate method
References
  informative
  normative
  overview
Refresh method
Relationship to other protocols
RemoveMissingDisk method
RemovePlex method
Repair method
Reset method

S

Security
Sequencing rules
  client
  server
Server
  abstract data model
  higher-layer triggered events
  initialization
  local events
  message processing
  overview
  sequencing rules
  timer events
  timers
Service
Service object
Service object interfaces
SetAllPathStatuses method
SetFileSystemFlags method
SetFlags method (section 3.3.5.2.4.15, section 3.3.5.2.18.6, section 3.3.5.2.25.10)
SetImportTarget method
SetInitiatorSharedSecret method
SetSANMode method
Shrink method (section 3.3.5.2.25.5, section 3.3.5.2.28.2)
Skip method
Software provider object interfaces
Standards assignments
Storage management objects (section 3.3.1.2, section 3.3.3.1)
Storage object relationships
Structures
  common data types
  IVdsAdvancedDisk
  IVdsAdvancedDisk2
  IVdsDisk
  IVdsHbaPort
  IVdsIscsiInitiatorAdapter
  IVdsIscsiInitiatorPortal
  IVdsPack
  IVdsProvider
  IVdsService
  IVdsServiceIscsi
  IVdsVolume
  IVdsVolumeMF
  IVdsVolumePlex
Syntax - message

T

Timer events
  client
  server
Timers
  client
  server
Transport - message
Triggered events - higher-layer - server

U

ULONGLONG
Unadvise method
UninstallDisks method

V

VDS client notifications
VDS sessions examples
VDS_ASYNC_OUTPUT structure
VDS_ASYNC_OUTPUT_TYPE enumeration
VDS_DISK_EXTENT structure
VDS_DISK_EXTENT_TYPE enumeration
VDS_DISK_FLAG enumeration
VDS_DISK_NOTIFICATION structure
VDS_DISK_PROP structure
VDS_DISK_STATUS enumeration
VDS_DRIVE_LETTER_FLAG enumeration
VDS_DRIVE_LETTER_NOTIFICATION structure
VDS_DRIVE_LETTER_PROP structure
VDS_FILE_SYSTEM_FLAG enumeration
VDS_FILE_SYSTEM_FORMAT_SUPPORT_FLAG enumeration
VDS_FILE_SYSTEM_FORMAT_SUPPORT_PROP structure
VDS_FILE_SYSTEM_NOTIFICATION structure
VDS_FILE_SYSTEM_PROP structure
VDS_FILE_SYSTEM_PROP_FLAG enumeration
VDS_FILE_SYSTEM_TYPE enumeration
VDS_FILE_SYSTEM_TYPE_PROP structure
VDS_HBAPORT_PROP structure
VDS_HBAPORT_SPEED_FLAG enumeration
VDS_HBAPORT_STATUS enumeration

VDS_HBAPORT_TYPE enumeration
VDS_HEALTH enumeration
VDS_INPUT_DISK structure
VDS_INTERCONNECT structure
VDS_INTERCONNECT_ADDRESS_TYPE enumeration
VDS_IPADDRESS structure
VDS_IPADDRESS_TYPE enumeration
VDS_ISCSI_INITIATOR_ADAPTER_PROP structure
VDS_ISCSI_INITIATOR_PORTAL_PROP structure
VDS_ISCSI_SHARED_SECRET structure
VDS_LUN_INFORMATION
VDS_LUN_INFORMATION structure
VDS_LUN_RESERVE_MODE enumeration
VDS_MOUNT_POINT_NOTIFICATION structure
VDS_NOTIFICATION structure
VDS_NOTIFICATION_TARGET_TYPE enumeration
VDS_OBJECT_TYPE enumeration
VDS_PACK_FLAG enumeration
VDS_PACK_NOTIFICATION structure
VDS_PACK_PROP structure
VDS_PACK_STATUS enumeration
VDS_PARTITION_FLAG enumeration
VDS_PARTITION_INFO_GPT structure
VDS_PARTITION_INFO_MBR structure
VDS_PARTITION_NOTIFICATION structure
VDS_PARTITION_PROP structure
VDS_PARTITION_STYLE enumeration
VDS_PATH_STATUS enumeration
VDS_PROVIDER_FLAG enumeration
VDS_PROVIDER_PROP structure
VDS_PROVIDER_TYPE enumeration
VDS_QUERY_PROVIDER_FLAG enumeration
VDS_REPARSE_POINT_PROP structure
VDS_SERVICE_FLAG enumeration
VDS_SERVICE_PROP structure
VDS_STORAGE_BUS_TYPE enumeration
VDS_STORAGE_DEVICE_ID_DESCRIPTOR structure
VDS_STORAGE_IDENTIFIER structure
VDS_STORAGE_IDENTIFIER_CODE_SET enumeration
VDS_STORAGE_IDENTIFIER_TYPE enumeration
VDS_TRANSITION_STATE enumeration
VDS_VOLUME_FLAG enumeration
VDS_VOLUME_NOTIFICATION structure
VDS_VOLUME_PLEX_PROP structure
VDS_VOLUME_PLEX_STATUS enumeration
VDS_VOLUME_PLEX_TYPE enumeration
VDS_VOLUME_PROP structure
VDS_VOLUME_STATUS enumeration
VDS_VOLUME_TYPE enumeration
VDS_WWN structure
Vendor-extensible fields
Versioning
Volume
  arrival
  removal
Volume object interfaces
Volume plex object interfaces
Volumes

W

Wait method
WaitForServiceReady method
Windows behavior