[MS-ISTM]: iSCSI Software Target Management Protocol

Revision Summary

Date	Revision History	Revision Class	Comments
08/27/2010	0.1	New	Released new document.
10/08/2010	0.2	Minor	Clarified the meaning of the technical content.
11/19/2010	0.3	Minor	Clarified the meaning of the technical content.
01/07/2011	0.3	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	0.3	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	0.3.1	Editorial	Changed language and formatting in the technical content.
05/06/2011	0.3.2	Editorial	Changed language and formatting in the technical content.
06/17/2011	0.4	Minor	Clarified the meaning of the technical content.
09/23/2011	0.4	No change	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	0.4	No change	No changes to the meaning, language, or formatting of the technical content.
03/30/2012	0.4	No change	No changes to the meaning, language, or formatting of the technical content.
07/12/2012	0.4	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	0.4	No change	No changes to the meaning, language, or formatting of the technical content.
01/31/2013	0.4	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1 Introduction

The **iSCSI Software Target** Management Protocol is specified in this document. It can set up **virtual disks**, **iSCSI targets**, and portals, configure **iSNS** hosting, schedule **snapshots**, and manage **resource groups** in a **high-availability (HA) cluster**. This protocol exposes its management operations through a set of **Component Object Model (COM)** interfaces [MS-DCOM].

The **Internet SCSI (iSCSI)** is an industry-standard protocol defined by [RFC3720]. It provides block-level storage access to the local **disks** of an iSCSI Software Target computer over a **TCP**/IP network from a remote location.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

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

authentication level
BLOB
Challenge-Handshake Authentication Protocol (CHAP)
class identifier (CLSID)
COM
Component Object Model (COM)
Coordinated Universal Time (UTC)
DCOM
disk
DNS
Domain Name System (DNS)
domain naming service name
dynamic disk
Dynamic Host Configuration Protocol (DHCP)
dynamic volume
fully qualified domain name (FQDN)
GUID
IDL
Interface Definition Language (IDL)
Internet Protocol security (IPsec)
Internet Protocol version 4 (IPv4)
Internet Protocol version 6 (IPv6)
Internet SCSI (iSCSI)
IPv4
IPv6
logical unit number (LUN)
mount point
multicast
mutual authentication
NetBIOS name
NT file system (NTFS)
NTFS

object class
page file or paging file
partition table
registry
remote procedure call (RPC)
RPC protocol sequence
SCSI
shadow copy
sparse file
system volume (SYSVOL)
Transmission Control Protocol (TCP)
unicast
Unicode
Universal Naming Convention (UNC)
universally unique identifier (UUID)
volume

The following terms are specific to this document:

activation: An operation that creates a new action instance.

burst length: The length, in bytes, of an asynchronous transmission of data from an iSCSI initiator to an iSCSI target.

high-availability (HA) cluster: An operational configuration of computers in which a pool of machines, called nodes, each one running an iSCSI Software Target instance, share configuration information, storage and networking resources, and provide seamless services to network-based clients, which are iSCSI initiators in the context of this specification.

Internet storage name service (iSNS): A naming service that is similar to DNS, but for storage networks.

iSCSI connection: A TCP connection, as defined in [RFC3720].

iSCSI initiator: A computer that requests access to a remote storage device. An iSCSI initiator issues small computer system interface (SCSI) commands to request services from components, which are logical units of a server known as "targets". For more information concerning iSCSI initiators and targets, see [RFC3720].

iSCSI initiator portal: The component of an iSCSI initiator that has a TCP/IP network address and that can 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 management application: An application that remotely configures and monitors storage devices via its connection to an iSCSI Software Target. The iSCSI management application typically presents a user interface that allows an administrator to initiate functions of the iSCSI Software Target Management Protocol.

iSCSI qualified name (IQN): A name that conforms to the format specified in [RFC3720] section 3.2.6.3.1.

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

iSCSI Software Target: A software implementation of the iSCSI technology specified in [RFC3720], along with supporting functionality.

iSCSI target: The entity that grants and facilitates access to a storage device, as specified in [RFC3720].

ISTM: The iSCSI Software Target Management Protocol.

ISTM client: The part of the iSCSI management application that remotely configures and monitors an iSCSI Software Target. The ISTM client runs on the client computer.

ISTM server: The part of the iSCSI Software Target that performs management functions on connections, sessions and devices in response to messages sent by the ISTM client. The ISTM server runs on the server computer.

medium access control (MAC): A data communication protocol sublayer that is part of the data-link layer of the seven-layer OSI model. It provides addressing and channel access control mechanisms that make it possible for several terminals or network nodes to communicate within a multipoint network, typically a local area network (LAN).

plug and play identifier: A resource that contains plug and play information for a hardware device component, which is required for a plug and play device driver.

resource group: An administrative mechanism to identify a logical collection of services and resources in a high-availability cluster environment, including configuration, storage, and networking, which are available to client computers. To a client, the services and resources appear to be offered by a single, continuously-available server, although each service or resource can be supplied by any server in the cluster. For the purposes of the iSCSI Software Target, a resource group acts as a scoping qualifier for enumerations and other operations. For example, when a resource group is specified in an ISTM method call, the information that is returned pertains to the resource group that the iSCSI Software Target computer is a part of, not necessarily to the iSCSI Software Target computer itself.

snapshot: A point-in-time, read-only copy of an iSCSI virtual disk.

target: A logical component of an iSCSI server that is an endpoint of an iSCSI transport interconnect. A target, also called a "host" in the context of this specification, provides services to iSCSI initiators. For more information, see [RFC3720].

VDS: The Virtual Disk Service (VDS) protocol [MS-VDS].

VHD file: A file that conforms to the virtual hard disk (VHD) file format. The ISTM implementation of the iSCSI Software Target uses VHD files to represent virtual disks. For more information, see [MSDN-VHD].

virtual disk: A disk that does not have a physical mechanical counterpart to it, and is not exposed as a hardware array LUN. It is a disk that uses a file to store its data. When this file is exposed to the operating system as a disk device, the exposed disk device emulates and, for all intents and purposes, behaves like a physical disk.Virtual disks can be implemented using a variety of formats including the VHD file format, the Universal Disk Format (UDF), and the format conforming to [ISO-9660] commonly called the "ISO file format", among others.

virtual hard disk (VHD): A file format, as defined by [MSFT-VHD].

volume shadow copy service (VSS): A service that coordinates the actions required to create a consistent snapshot of backup data without affecting the running of the application that is using the data.

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

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

1.2.1 Normative References

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

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

[MS-CMRP] Microsoft Corporation, "Failover Cluster: Management API (ClusAPI) Protocol".

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

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

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

[MS-OAUT] Microsoft Corporation, "OLE Automation Protocol".

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

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

[RFC1001] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", STD 19, RFC 1001, March 1987, http://www.ietf.org/rfc/rfc1001.txt

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989, http://www.ietf.org/rfc/rfc1123.txt

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

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

[RFC4171] Tseng, J., Gibbons, K., Travostino, F., et al., "Internet Storage Name Service (iSNS)", RFC 4171, September 2005, http://www.ietf.org/rfc/rfc4171.txt

[SAM-3] International Committee on Information Technology Standards, "SCSI Architecture Model - 3 (SAM-3)", September 2004, http://www.t10.org/cgi-bin/ac.pl?t=f&f=sam3r14.pdf

1.2.2 Informative References

[ANSI-289-1996] American National Standards Institute, "Fibre Channel - Fabric Generic Requirements (FC-FG)", ANSI INCITS 289-1996 (R2001), 2001, http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+INCITS+289-1996+(R2001)

Note There is a charge to download the specification.

[IANAIPV6A] IANA, "IPV6 Addressing", April 2010, http://msdn.microsoft.com/en-us/library/aa917150.aspx

[IANAIPV6AS] IANA, "Internet Protocol Version 6 Address Space", March 2010, http://www.iana.org/assignments/ipv6-address-space

[IEEE802.1X] Institute of Electrical and Electronics Engineers, "IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control", December 2004, http://ieeexplore.ieee.org/iel5/9828/30983/01438730.pdf

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

[MSDN-IStream] Microsoft Corporation, "IStream interface", http://msdn.microsoft.com/en-us/library/aa380034.aspx

[MSDN-VHD] Microsoft Corporation, "About Virtual Hard Disk", http://msdn.microsoft.com/en-us/library/dd323654(VS.85).aspx

[MSFT-VHD] Microsoft Corporation, "VHD Specification", February 2009, http://www.microsoft.com/downloads/details.aspx?displaylang=en&familyid=c2d03242-2ffb-48ef-a211-f0c44741109e

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

[RFC1994] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996, http://www.ietf.org/rfc/rfc1994.txt

[RFC3980] Krueger, M. and Elliott R., "T11 Network Address Authority (NAA) Naming Format for iSCSI Node Names", RFC 3980, February 2005, http://www.ietf.org/rfc/rfc3980.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/cgi-bin/ac.pl?t=f&f=/spc3r23.pdf

1.3 Overview

The iSCSI Software Target Management Protocol supports disk management from an administrative command line tool or user interface tool. At a high level, an Internet SCSI (iSCSI) [RFC3720] environment consists of an **iSCSI initiator**, which is the entity that requests access to a storage device, and an iSCSI target, which is the entity that grants and facilitates the requested access. The iSCSI target is typically surrounded by supporting hardware or software functionality. In this specification, the iSCSI target implementation along with its supporting functionality is called the iSCSI Software Target.

A high-level view of these concepts is shown in the following diagram.

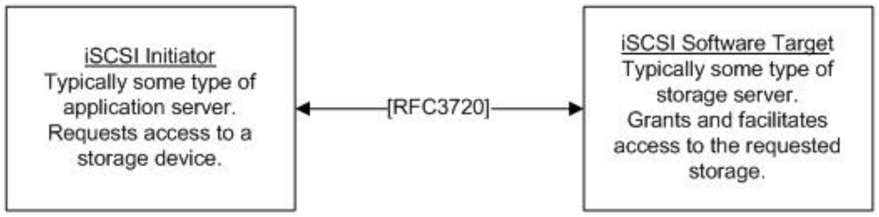

iSCSI Initiator
Typically some type of application server. Requests access to a storage device.
[RFC3720]
iSCSI Software Target
Typically some type of storage server. Grants and facilitates access to the requested storage.

Figure 1: An iSCSI server environment

The iSCSI Software Target itself can be viewed as containing two components: the iSCSI target, as defined in [RFC3720], and the **ISTM server**, which is the server side of the iSCSI Software Target Management Protocol.

Many processes of the iSCSI Software Target require configuration and monitoring. This can be performed remotely by an **iSCSI management application** using ISTM, as shown in the following diagram.

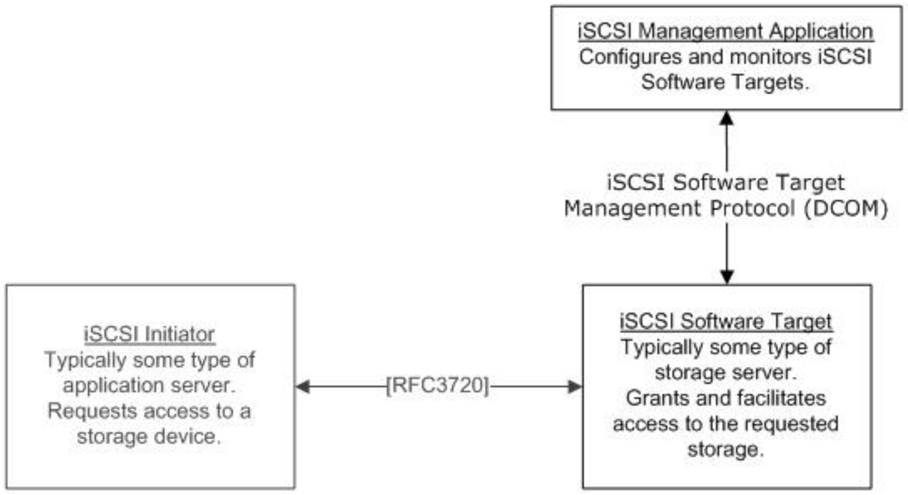

iSCSI Management Application
Configures and monitors iSCSI Software Targets.
iSCSI Software Target Management Protocol (DCOM)
iSCSI Initiator
Typically some type of application server. Requests access to a storage device.
[RFC3720]
iSCSI Software Target
Typically some type of storage server. Grants and facilitates access to the requested storage.

Figure 2: An iSCSI environment with remote management

ISTM defines a client role, the **ISTM client**, and a server role, the ISTM server, as indicated by the following general sequence diagram. The iSCSI management application usually performs the client role, and the iSCSI Software Target usually performs the server role.

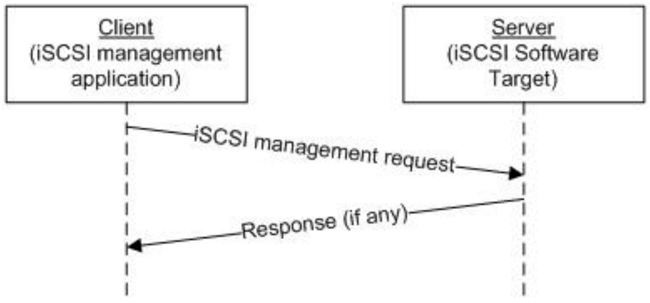


Figure 3: General request/response sequence for the ISTM Protocol

More detailed sequence diagrams that correspond to specific areas of protocol behavior are shown in subsequent sections, as appropriate.

A common operation that the iSCSI management application performs is the enumeration of various objects that exist on the iSCSI Software Target. These objects include sessions, connections, **logical unit numbers (LUNs)**, and snapshots, among others. See the sections that follow for a complete list.

The sequence of messages that is used to perform such enumerations takes a common form, as shown in the following diagram.

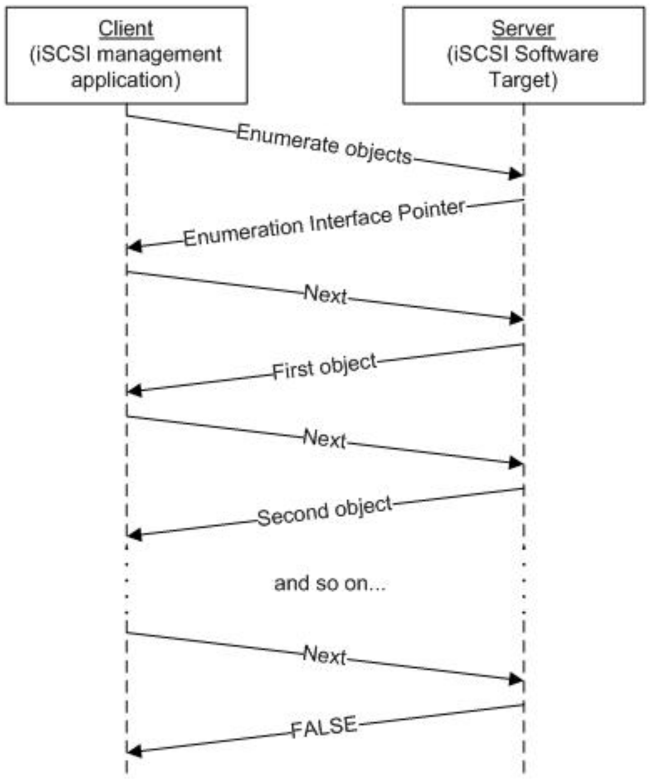


Figure 4: General enumeration sequence for the ISTM Protocol

The remote management capabilities provided by the ISTM protocol are implemented as **COM** interfaces that expose COM objects. These COM objects correspond to the components and functionality of the iSCSI Software Target, which can be grouped into the following subsystems:

- Feature Management
- Session Management
- Local Device Management
- LUN Management
- Snapshot Management
- iSNS Management

- **Target** Management

The subsections that follow provide additional information on each of these subsystems.

1.3.1 Feature Management

The feature management subsystem applies to settings and statuses that affect multiple components of the software; for the purposes of this specification, these settings and statuses will be called "global". Global settings and statuses include service identification, licensing, snapshot considerations, client access verification, registration for status notification, and membership in a high-availability (HA) cluster configuration. See section 3.1.1.1 for details about the global settings and statuses that are available to ISTM.

The sequence of messages for feature management follows the simple request/response model described in section 1.3 with the exception of the registration for status notification. This feature is more complex because it involves the creation of a COM server and COM interface on the client computer for use by the ISTM server when sending unsolicited status information. This activity is captured in the following sequence diagram.

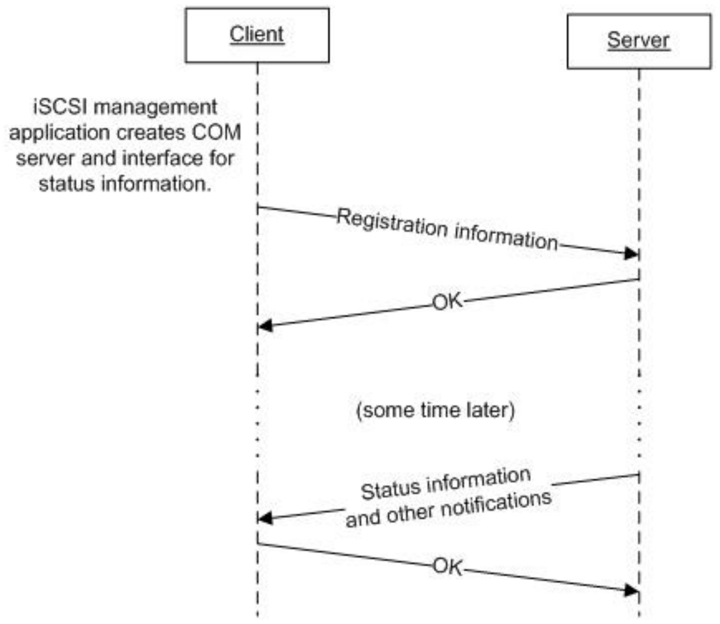

Client
Server
iSCSI management application creates COM server and interface for status information.
Registration information
OK
(some time later)
Status information and other notifications
OK

Figure 5: Registering for and receiving status notifications

More detailed information about this operation is provided in sections 3.41 and 3.42.

1.3.2 Session Management

The session management subsystem allows an iSCSI management application to enumerate all existing connections and sessions, as defined in [RFC3720]. The sequence of messages for such enumerations follows the general enumeration sequence diagram shown in section 1.3.

The iSCSI Software Target also allows an iSCSI management application to retrieve the details of a specific connection or session. The sequence of messages for these operations follows the general sequence diagram shown in section 1.3.

More detailed information about these operations is provided in section 3.1.1.2.

1.3.3 Local Device Management

The local device management subsystem involves a large number of configuration and inquiry operations that allow such activities as system discovery and detailed disk-information gathering.

Enumeration operations

These operations are for the enumeration of objects. The sequence of messages for such enumerations follows the general enumeration sequence diagram shown in section 1.3.

- Enumerate all **volumes** in the iSCSI Software Target computer.
- Enumerate the directories that exist in a volume.
- Enumerate the portals (as defined in [RFC3720]) that exist on the iSCSI Software Target.
- Enumerate the **resource groups** that relate to the HA clustering configuration of the iSCSI Software Target computer.

Creation, deletion, and configuration operations

These operations are for the creation, deletion, or some type of configuration of an object. The sequence of messages for these operations follows the general sequence diagram shown in section 1.3.

- Create a **VHD file** that represents a virtual disk.
- Increase the size of a VHD file.
- Delete a VHD file.
- Configure a particular portal.
- Request that the iSCSI Software Target refresh its device information.
- Revise information about a particular resource group.
- Create a differencing VHD file.

Informational operations

These operations are used to obtain information about the system or about a particular object. The sequence of messages for these operations follows the general sequence diagram shown in section 1.3.

- Determine whether a specified volume is suitable to host a VHD file.
- Determine whether the iSCSI Software Target computer can partition disks.
- Determine whether the specified VHD file is valid.
- Determine whether the volume being accessed is an **NT file system (NTFS)** volume.
- Determine whether the specified VHD file exists.
- Retrieve information about a particular portal.

- Retrieve information about the physical processors that are available to run the iSCSI Software Target software.
- Retrieve information about a particular volume.
- Retrieve information about a particular resource group.

More detailed information about these operations is provided in section 3.1.1.3.

1.3.4 LUN Management

The LUN management subsystem is used to create and manage LUNs, which represent the disks from the point of view of the initiator. Certain management operations assume the point of view of the server running the iSCSI Software Target. In this section, the term "virtual disk" is used to indicate the iSCSI target point of view.

An ISTM client can perform the following disk management operations:

- Enumerate all virtual disks that reside on the same volume as a specified virtual disk.
- Create or add a new disk.
- Create or add a new disk by using a snapshot,
- Open a disk with settings including serial number and disk index.
- Get a specific disk.
- Surface a virtual disk as a physical disk.
- Delete a specific disk.

On a specific disk, an ISTM client can perform the following disk management operations:

- Query the state, including the mount status, mount path, and current device id.
- Check if the disk is accessible.
- Save the settings.
- Surface the disk for read/write locally at the machine running the iSCSI Software Target.
- Un-mount the disk.
- Enumerate **mount points** associated with a virtual disk when seen as a physical disk.

More detailed information about these operations is provided in section 3.1.1.4.

1.3.5 Snapshot Management

The snapshot management subsystem determines how a disk restore operation is performed from a backup. It interfaces with a physical backup utility.

An ISTM client can perform the following snapshot-management operations:

- Create a snapshot for a given LUN.
- Delete a snapshot for a given LUN.

- Create a LUN to match the state of a snapshot of an original LUN.
- Surface the snapshot as a physical disk that is locally read-only on the machine running the iSCSI Software Target.
- Un-mount the snapshot.

More detailed information about these operations is provided in section 3.1.1.5.

1.3.6 iSNS Management

The Internet storage name service (iSNS) is a naming service that is similar to the **Domain Name System (DNS)**, but for storage networks. It manages the relationship between a target and a physical device. The storage devices register their names with the manager, which differs from **DNS**. For more information, see [RFC4171].

An ISTM client can perform the following iSNS management operations:

- Enumeration of existing iSNS servers.
- Registration of the iSCSI Software Target towards a given iSNS server.
- Un-registration of the iSCSI Software Target from a given iSNS server.

More detailed information about these operations is provided in section 3.1.1.6.

1.3.7 Target Management

The target management subsystem manages information about the iSCSI targets that initiators can connect to, along with mappings to disks. It supports target management operations including creating or adding iSCSI targets, assigning a virtual disk to a target, removing a virtual disk from a target, and deleting a target.

An ISTM client can perform the following operations:

- Add or create an iSCSI target by providing identification for the target in the form of an **iSCSI qualified name (IQN)** or **domain naming service name**. Additionally the client can specify a resource group for an HA cluster environment.
- Get a specific target by providing identification of the target.
- Open a target with specified target settings like name or description.
- Enumerate all the targets in the server .
- Delete a specific target by providing identification of the target.

The following sequence diagrams depict the preceding scenarios:

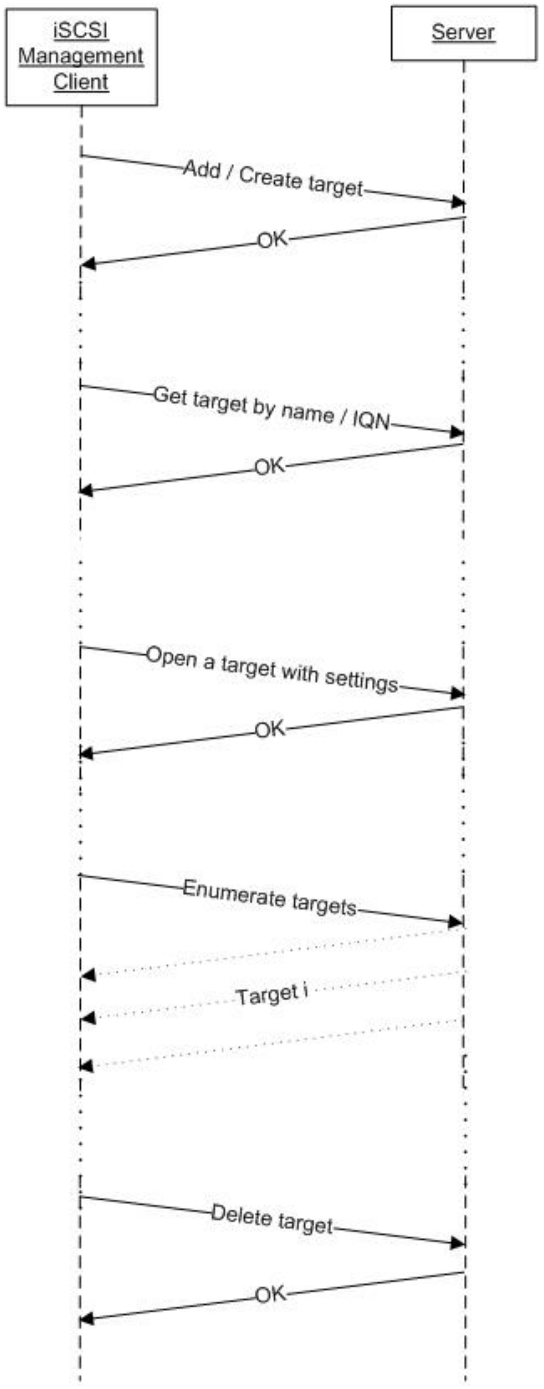


Figure 6: Target management sequences

Once a target is added, the ISTM client can perform the following operations on this target:

- Enumerate identification methods for this target.
- Add a disk.

- Remove a specific disk or all the disks.
- Set the LUN mappings for a specific disk.
- Enumerate attached disks.
- Add or remove a portal.
- Enumerate all the portals servicing storage requests for this target.
- Save settings.
- While in an HA cluster configuration, change the target state from online to offline and vice versa.

More detailed information about these operations is provided in section 3.1.1.7.

1.4 Relationship to Other Protocols

The iSCSI Software Target Management Protocol relies on **DCOM** [MS-DCOM], which uses **remote procedure call (RPC)** [MS-RPCE] as its transport. The transport layers for this protocol are shown in the following diagram:

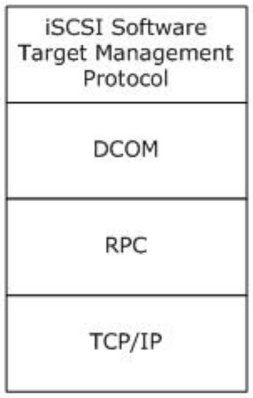


Figure 7: iSCSI Software Target Management Protocol Transport Layers

No protocol depends on ISTM.

1.5 Prerequisites/Preconditions

The iSCSI Software Target Management Protocol is implemented over DCOM and RPC, and it has the prerequisites specified in [MS-DCOM], [MS-OAUT], and [MS-RPCE] that are common to DCOM, OLE automation, and RPC interfaces, respectively.

An iSCSI target and iSCSI initiator that comply with RFC 3720, "Internet Small Computer Systems Interface (iSCSI)" ([RFC3720]), are required to be present on the operating system on which an implementation of this protocol runs.

1.6 Applicability Statement

The iSCSI Software Target Management Protocol is applicable to centralized, software-based and hardware-independent iSCSI disk subsystems in storage area networks (SANs). The ISTM protocol can be used to create and manage iSCSI targets and iSCSI virtual disks.

ISTM is a scalable solution designed to enable customers to quickly install and configure a full-featured storage solution, as well as to manage other storage already deployed in their environment. It can support a broad range of file-serving protocols, making file sharing possible across heterogeneous environments including UNIX, Novell, Apple, and Windows. See section 1.3 for additional information concerning ISTM management capabilities.

1.7 Versioning and Capability Negotiation

Supported Transports: The iSCSI Software Target Management Protocol uses the Distributed Component Object Model (DCOM) Remote Protocol [MS-DCOM], which in turn uses RPC over TCP, as its only transport. See section 2.1 for details.

Protocol Version: This protocol consists of DCOM interfaces, all of which are version 1.0.

Functionality Negotiation: This protocol uses return codes as a capability discovery mechanism; the server returns **E_NOTIMPL** to indicate that a function is not implemented.

Security and Authentication Methods: This protocol relies on the security and authentication provided by DCOM and by Remote Procedure Call Protocol Extensions [MS-RPCE] and configures it as specified in section 2.1.

Localization: This protocol supports no locale-specific or localized resources.

1.8 Vendor Extensible Fields

The iSCSI Software Target Management Protocol does not define any vendor-extensible fields.

Unless otherwise specified, all ISTM methods MUST return either zero to indicate success or a non-error **HRESULT** (section 2.2) value [MS-ERREF] or implementation-specific error code to indicate failure.

HRESULT values are vendor-extensible. Vendors can choose their own values for **HRESULT**; however, the C bit (0x20000000) MUST be set, indicating a vendor-extended code, as specified in [MS-ERREF].

1.9 Standards Assignments

The following table lists the **CLSID** values that are used by an ISTM client to access the ISTM server interfaces that correspond to the management subsystems described in section 1.3. To access an interface, a client uses DCOM **activation** with the interface name and the CLSID of the **object class** that supports the interface.

Subsystem	Interface Name	CLSID Value	Reference
Feature Management	IWTGeneral	baa4a577-202d-49c7-a6bb-10ee914ee199	section 3.52
Session Management	ISessionManager	ad069971-edc5-4357-a132-430dad479d2b	section 3.34
Local Device Management	ILocalDeviceMgr	f1d3a46c-2e1b-4d84-97da-b3742ad67871	section 3.30
LUN Management	IWTDiskMgr	0144084b-e09e-4f45-a56b-dcdc9f379f5e	section 3.46

Subsystem	Interface Name	CLSID Value	Reference
Snapshot Management	ISnapshotMgr	4e7645fa-1a95-416c-b38f-ad574a3c3e57	section 3.38
iSNS Management	ISnsMgr	24f9c1a9-b22e-4e24-bec9-4af0a4d96736	section 3.40
Target Management	IHostMgr	e14efe0b-fd3b-41ea-8e3b-88930445b944	section 3.25

The following table lists the **universally unique identifier (UUID)** values for all the interfaces specified in section 3.

Parameter	Value	Reference
UUID for IDirectoryEnum	28bc8d5e-ca4b-4f54-973c-ed9622d2b3ac	[C706]
UUID for IEnumCachedInitiator	f093fe3d-8131-4b73-a742-ef54c20b337b	[C706]
UUID for IEnumConnection	6aea6b26-0680-411d-8877-a148df3087d5	[C706]
UUID for IEnumDisk	bb39e296-ad26-42c5-9890-5325333bb11e	[C706]
UUID for IEnumDisk2	a5ecfc73-0013-4a9e-951c-59bf9735fdda	[C706]
UUID for IEnumHost	e141fd54-b79e-4938-a6bb-d523c3d49ff1	[C706]
UUID for IEnumIDMethod	345b026b-5802-4e38-ac75-795e08b0b83f	[C706]
UUID for IEnumLMMountPoint	100da538-3f4a-45ab-b852-709148152789	[C706]
UUID for IEnumPortal	1995785d-2a1e-492f-8923-e621eaca39d9	[C706]
UUID for IEnumResourceGroup	640038f1-d626-40d8-b52b-09660601d045	[C706]
UUID for IEnumSession	40cc8569-6d23-4005-9958-e37f08ae192b	[C706]
UUID for IEnumSession2	442931d5-e522-4e64-a181-74e98a4e1748	[C706]
UUID for IEnumSnapshot	66c9b082-7794-4948-839a-d8a5a616378f	[C706]
UUID for IEnumSnsServer	e2842c88-07c3-4eb0-b1a9-d3d95e76fef2	[C706]
UUID for IEnumVolume	81fe3594-2495-4c91-95bb-eb5785614ec7	[C706]
UUID for IEnumVolume2	8c58f6b3-4736-432a-891d-389de3505c7c	[C706]
UUID for IEnumWTDisk	56e65ea5-cdff-4391-ba76-006e42c2d746	[C706]
UUID for IEnumWTDiskLunMapping	1396de6f-a794-4b11-b93f-6b69a5b47bae	[C706]
UUID for IHost	b0076fec-a921-4034-a8ba-090bc6d03bde	[C706]
UUID for IHost2	b4fa8e86-2517-4a88-bd67-75447219eee4	[C706]
UUID for IHost3	fe7f99f9-1dfb-4afb-9d00-6a8dd0aabf2c	[C706]
UUID for IHostMgr	b06a64e3-814e-4ff9-afac-597ad32517c7	[C706]

Parameter	Value	Reference
UUID for IHostMgr2	dd6f0a28-248f-4dd3-afe9-71aed8f685c4	[C706]
UUID for IHostMgr3	01454b97-c6a5-4685-bea8-9779c88ab990	[C706]
UUID for ILocalDeviceMgr	8ad608a4-6c16-4405-8879-b27910a68995	[C706]
UUID for ILocalDeviceMgr2	b0d1ac4b-f87a-49b2-938f-d439248575b2	[C706]
UUID for ILocalDeviceMgr3	e645744b-cae5-4712-acaf-13057f7195af	[C706]
UUID for ISessionManager	8d7ae740-b9c5-49fc-a11e-89171907cb86	[C706]
UUID for ISessionManager2	c10a76d8-1fe4-4c2f-b70d-665265215259	[C706]
UUID for ISnapshot	883343f1-ceed-4e3a-8c1b-f0dadfce281e	[C706]
UUID for ISnapshotMgr	4e65a71e-4ede-4886-be67-3c90a08d1f29	[C706]
UUID for ISnsMgr	1b1c4d1c-abc4-4d3a-8c22-547fba3aa8a0	[C706]
UUID for IStatusNotify	312cc019-d5cd-4ca7-8c10-9e0a661f147e	[C706]
UUID for IWTDisk	866a78bc-a2fb-4ac4-94d5-db3041b4ed75	[C706]
UUID for IWTDisk2	348a0821-69bb-4889-a101-6a9bde6fa720	[C706]
UUID for IWTDisk3	1822a95e-1c2b-4d02-ab25-cc116dd9dbde	[C706]
UUID for IWTDiskMgr	52ba97e7-9364-4134-b9cb-f8415213bdd8	[C706]
UUID for IWTDiskMgr2	592381e5-8d3c-42e9-b7de-4e77a1f75ae4	[C706]
UUID for IWTDiskMgr3	d6bd6d63-e8cb-4905-ab34-8a278c93197a	[C706]
UUID for IWTDiskMgr4	703e6b03-7ad1-4ded-ba0d-e90496ebc5de	[C706]
UUID for IWTGeneral	d71b2cae-33e8-4567-ae96-3ccf31620be2	[C706]
UUID for IWTGeneral2	3c73848a-a679-40c5-b101-c963e67f9949	[C706]

2 Messages

2.1 Transport

The iSCSI Software Target Management Protocol uses the DCOM Remote Protocol, as specified in [MS-DCOM], as its transport. On its behalf, the DCOM Remote Protocol uses the TCP/IP **RPC protocol sequence**, as defined in [MS-RPCE] section 2.1.1.1.

The ISTM interfaces make use of the underlying DCOM security framework, as specified in [MS-DCOM]. To access an interface, an ISTM client uses DCOM activation with the interface name and the CLSID of the object class that supports the interface. The CLSID values are defined in section 1.9.

2.2 Common Data Types

In addition to RPC base types and definitions specified in [C706] and [MS-RPCE], additional data types are defined in this section.

The following data types are specified in [MS-DTYP]:

Data type name	Section	Description
BOOL	2.2.3	A true or false value. **Note** Unless otherwise specified, Boolean values corresponding to this data type MUST be either 0 (false) or 1 (true).
BYTE	2.2.6	An 8-bit unsigned value
DWORD	2.2.9	A 32-bit unsigned integer.
DWORDLONG	2.2.13	A 64-bit unsigned integer.
FILETIME	2.3.3	A 64-bit **Coordinated Universal Time (UTC)** value. **Note** Unless otherwise specified, **FILETIME** values express the number of 100 nanosecond units that have elapsed since January 1, 1601.
GUID	2.3.4	A unique identifier for an object.
HRESULT	2.2.18	An error or warning value.
ULONG	2.2.51	A 32-bit unsigned integer.
WCHAR	2.2.60	A 16-bit **Unicode** character.
WORD	2.2.61	A 16-bit unsigned integer.

The following data types are specified in [MS-VDS]:

Data type name	Section	Description
VDS_LUN_INFORMATION	2.2.1.3.16	A structure that contains information about a **VHD disk** LUN.
VDS_STORAGE_BUS_TYPE	2.2.1.2.5	An enumeration that defines the type of bus on

Data type name	Section	Description
		which a disk resides.
VDS_STORAGE_DEVICE_ID_DESCRIPTOR	2.2.1.3.14	A structure that contains information about a device identification descriptor.
VDS_STORAGE_IDENTIFIER	2.2.1.3.13	A structure that contains information about a storage identifier.
VDS_STORAGE_IDENTIFIER_CODE_SET	2.2.1.2.6	An enumeration that defines the type of bus on which a disk resides.
VDS_STORAGE_IDENTIFIER_TYPE	2.2.1.2.7	An enumeration that defines the types of storage device identifiers, as described in [SPC-3].

The following data types are specified in [MS-OAUT]:

Data type name	Section	Description
BSTR	2.2.23.2	An OLE automation type for transferring length-prefixed strings. **Note** Unless otherwise specified, when an IP address is contained by a **BSTR** variable, the variable contains the string representation of the IP address in dotted-decimal notation for **Internet Protocol version 4 (IPv4)** addresses and in colon-hexadecimal notation for **Internet Protocol version 6 (IPv6)** addresses. For information on **IPv6** addressing, see [IANAIPV6A].<1>

2.2.1 Common Error Codes

The following **HRESULT** (section 2.2) return codes can be used by this protocol, in addition to the **HRESULT** return codes specified in [MS-ERREF]. These codes SHOULD be returned by the ISTM server to provide additional information about the result of a method call or about the reason a call failed. If the result is an error rather than simple status information, the most significant bit of the **HRESULT** MUST be set, as specified in [MS-ERREF].

Error code / value	Description
S_FALSE 0x00000001	The operation completed logically with a false outcome.
S_OK 0x00000000	The operation completed successfully.
VSS_E_ASRERROR_CRITICAL_DISK_CANNOT_BE_EXCLUDED 0x80042415	Restore failed because a disk which was critical at backup is excluded.
VSS_E_ASRERROR_CRITICAL_DISKS_TOO_SMALL 0x80042408	Failed to find enough suitable disks for recreating all critical

Error code / value	Description
	disks. The number of available disks should be same or greater than the number of critical disks at time of backup, and each one of the disks must be of same or greater size.
VSS_E_ASRERROR_DATADISK_RDISK0 0x80042406	A data disk is currently set as active in BIOS. Set some other disk as active or use the DiskPart utility to clean the data disk, and then retry the restore operation.
VSS_E_ASRERROR_DISK_ASSIGNMENT_FAILED 0x80042401	There are too few disks on this computer or one or more of the disks is too small. Add or change disks so they match the disks in the backup, and try the restore again.
VSS_E_ASRERROR_DISK_RECREATION_FAILED 0x80042402	Windows cannot create a disk on this computer needed to restore from the backup. Make sure the disks are properly connected, or add or change disks, and try the restore again.
VSS_E_ASRERROR_DYNAMIC_VHD_NOT_SUPPORTED 0x8004240A	A critical dynamic disk is a Virtual Hard Disk (VHD). This is an unsupported configuration.
VSS_E_ASRERROR_FIXED_PHYSICAL_DISK_AVAILABLE_AFTER_DISK_EXCLUSION 0x80042414	Windows did not find any disk which it can use for recreating volumes present in backup.
VSS_E_ASRERROR_MISSING_DYNDISK 0x80042404	The backup failed due to a missing disk for a dynamic volume. Ensure the

Error code / value	Description
	disk is online and retry the backup.
VSS_E_ASRERROR_NO_ARCPATH 0x80042403	The computer needs to be restarted to finish preparing a hard disk for restore. To continue, restart your computer and run the restore again.
VSS_E_ASRERROR_NO_PHYSICAL_DISK_AVAILABLE 0x80042413	Windows did not find any fixed disk that can be used to recreate volumes present in backup.
VSS_E_ASRERROR_RDISK_FOR_SYSTEM_DISK_NOT_FOUND 0x80042412	No disk that can be used for recovering the system disk can be found.
VSS_E_ASRERROR_RDISK0_TOOSMALL 0x80042407	The disk that is set as active in BIOS is too small to recover the original system disk. Replace the disk with a larger one and retry the restore operation.
VSS_E_ASRERROR_SHARED_CRIDISK 0x80042405	Automated System Recovery failed the shadow copy, because a selected critical volume is located on a cluster shared disk. This is an unsupported configuration.
VSS_E_ASRERROR_SYSTEM_PARTITION_HIDDEN 0x80042416	System partition (partition marked active) is hidden or contains an unrecognized file system.
VSS_E_AUTORECOVERY_FAILED 0x800423FB	The autorecovery operation failed to complete on the shadow copy.
VSS_E_BAD_STATE 0x80042301	A function call was made when the object was in an incorrect state.

Error code / value	Description
VSS_E_BREAK_REVERT_ID_FAILED 0x800423F6	The shadow copy set break operation failed because the disk/partition identities could not be reverted. The target identity already exists on the machine or cluster and must be masked before this operation can succeed.
VSS_E_CANNOT_REVERT_DISKID 0x800423FE	The MBR signature or GPT ID for one or more disks could not be set to the intended value. Check the Application event log for more information.
VSS_E_CLUSTER_ERROR 0x80042400	The clustered disks could not be enumerated or could not be put into cluster maintenance mode. Check the System event log for cluster related events and the Application event log for VSS related events.
VSS_E_CLUSTER_TIMEOUT 0x8004232E	A timeout occurred while preparing a cluster shared volume for backup.
VSS_E_CORRUPT_XML_DOCUMENT 0x80042310	The given XML document is invalid. It is either incorrectly-formed XML or it does not match the schema.
VSS_E_CRITICAL_VOLUME_ON_INVALID_DISK 0x80042411	A critical volume selected for backup exists on a disk which cannot be backed up by ASR.
VSS_E_DYNAMIC_DISK_ERROR 0x800423FC	An error occurred in processing the dynamic disks

Error code / value	Description
	involved in the operation.
VSS_E_FLUSH_WRITES_TIMEOUT 0x80042313	The shadow copy provider timed out while flushing data to the volume being shadow copied. This is probably due to excessive activity on the volume. Try again later when the volume is not being used so heavily.
VSS_E_HOLD_WRITES_TIMEOUT 0x80042314	The shadow copy provider timed out while holding writes to the volume being shadow copied. This is probably due to excessive activity on the volume by an application or a system service. Try again later when activity on the volume is reduced.
VSS_E_INSUFFICIENT_STORAGE 0x8004231F	Insufficient storage available to create either the shadow copy storage file or other shadow copy data.
VSS_E_INVALID_XML_DOCUMENT 0x80042311	The given XML document is invalid. It is either incorrectly-formed XML or it does not match the schema.
VSS_E_LEGACY_PROVIDER 0x800423F7	This version of the hardware provider does not support this operation.
VSS_E_MAXIMUM_DIFFAREA_ASSOCIATIONS_REACHED 0x8004231E	Maximum number of shadow copy storage associations already reached.
VSS_E_MAXIMUM_NUMBER_OF_REMOTE_MACHINES_REACHED 0x80042322	The maximum number of remote machines for this operation has been reached.

Error code / value	Description
VSS_E_MAXIMUM_NUMBER_OF_SNAPSHOTS_REACHED 0x80042317	The specified volume has already reached its maximum number of shadow copies.
VSS_E_MAXIMUM_NUMBER_OF_VOLUMES_REACHED 0x80042312	The maximum number of volumes for this operation has been reached.
VSS_E_MISSING_DISK 0x800423F8	An expected disk did not arrive in the system.
VSS_E_MISSING_HIDDEN_VOLUME 0x800423F9	An expected hidden volume did not arrive in the system. Check the Application event log for more information.
VSS_E_MISSING_VOLUME 0x800423FA	An expected volume did not arrive in the system. Check the Application event log for more information.
VSS_E_NESTED_VOLUME_LIMIT 0x8004232C	The specified volume is nested too deeply to participate in the VSS operation.
VSS_E_NO_SNAPSHOTS_IMPORTED 0x80042320	No shadow copies were successfully imported.
VSS_E_NONTRANSPORTABLE_BCD 0x800423FD	The given Backup Components Document is for a non-transportable shadow copy. This operation can only be done on transportable shadow copies.
VSS_E_NOT_SUPPORTED 0x8004232F	The requested operation is not supported.
VSS_E_OBJECT_ALREADY_EXISTS 0x8004230D	The object already exists.
VSS_E_OBJECT_NOT_FOUND	The specified object

Error code / value	Description
0x80042308	was not found.
VSS_E_PROVIDER_ALREADY_REGISTERED 0x80042303	The provider has already been registered.
VSS_E_PROVIDER_IN_USE 0x80042307	The shadow copy provider is currently in use and cannot be unregistered.
VSS_E_PROVIDER_NOT_REGISTERED 0x80042304	The volume shadow copy provider is not registered in the system.
VSS_E_PROVIDER_VETO 0x80042306	The shadow copy provider had an error.
VSS_E_REBOOT_REQUIRED 0x80042327	A reboot is required after completing this operation.
VSS_E_REMOTE_SERVER_UNAVAILABLE 0x80042323	The remote server is unavailable.
VSS_E_REMOTE_SERVER_UNSUPPORTED 0x80042324	The remote server is running a version of the Volume Shadow Copy Service that does not support remote shadow-copy creation.
VSS_E_RESYNC_IN_PROGRESS 0x800423FF	The LUN resynchronization operation could not be started because another resynchronization operation is already in progress.
VSS_E_REVERT_IN_PROGRESS 0x80042325	A revert is currently in progress for the specified volume. Another revert cannot be initiated until the current revert completes.
VSS_E_REVERT_VOLUME_LOST 0x80042326	The volume being reverted was lost during revert.
VSS_E_SNAPSHOT_NOT_IN_SET	The shadow copy ID was not found in the

Error code / value	Description
0x8004232B	backup components document for the shadow copy set.
VSS_E_SNAPSHOT_SET_IN_PROGRESS 0x80042316	Another shadow copy creation is already in progress. Wait a few moments and try again.
VSS_E_SOME_SNAPSHOTS_NOT_IMPORTED 0x8004232	Some shadow copies were not successfully imported.
VSS_E_TRANSACTION_FREEZE_TIMEOUT 0x80042328	A timeout occurred while freezing a transaction manager.
VSS_E_TRANSACTION_THAW_TIMEOUT 0x80042329	Too much time elapsed between freezing a transaction manager and thawing the transaction manager.
VSS_E_UNEXPECTED 0x80042302	A Volume Shadow Copy Service component encountered an unexpected error.
VSS_E_UNEXPECTED_PROVIDER_ERROR 0x8004230F	The shadow copy provider had an unexpected error while trying to process the specified operation.
VSS_E_UNEXPECTED_WRITER_ERROR 0x80042315	VSS encountered problems while sending events to writers.
VSS_E_UNSELECTED_VOLUME 0x8004232A	The requested operation would overwrite a volume that is not explicitly selected. For more information, check the Application event log.
VSS_E_UNSUPPORTED_CONTEXT 0x8004231B	The shadow copy provider does not support the specified

Error code / value	Description
	shadow copy type.
VSS_E_VOLUME_IN_USE 0x8004231D	The specified shadow copy storage association is in use and so can't be deleted.
VSS_E_VOLUME_NOT_LOCAL 0x8004232D	The volume being backed up is not mounted on the local host.
VSS_E_VOLUME_NOT_SUPPORTED 0x8004230C	Shadow copying the specified volume is not supported.
VSS_E_VOLUME_NOT_SUPPORTED_BY_PROVIDER 0x8004230E	The given shadow copy provider does not support shadow copying the specified volume.
VSS_E_WRITER_ALREADY_SUBSCRIBED 0x8004231A	The writer has already successfully called the Subscribe function. It cannot call Subscribe multiple times.
VSS_E_WRITER_INFRASTRUCTURE 0x80042318	An error was detected in the Volume Shadow Copy Service (VSS). The problem occurred while trying to contact VSS writers.
VSS_E_WRITER_NOT_RESPONDING 0x80042319	A writer did not respond to a GatherWriterStatus call. The writer may either have terminated or it may be stuck.
VSS_E_WRITER_STATUS_NOT_AVAILABLE 0x80042409	Writer status is not available for one or more writers. A writer may have reached the limit to the number of available backup-restore session states.
VSS_E_WRITERERROR_INCONSISTENTSNAPSHOT	The shadow-copy set

Error code / value	Description
0x800423F0	only contains only a subset of the volumes needed to correctly backup the selected components of the writer.
VSS_E_WRITERERROR_NONRETRYABLE 0x800423F4	The writer experienced a non-transient error. If the backup process is retried, the error is likely to reoccur.
VSS_E_WRITERERROR_OUTOFRESOURCES 0x800423F1	A resource allocation failed while processing this operation.
VSS_E_WRITERERROR_PARTIAL_FAILURE 0x80042336	The writer experienced a partial failure. Check the component level error state for more information.
VSS_E_WRITERERROR_RECOVERY_FAILED 0x800423F5	The writer experienced an error while trying to recover the shadow-copy volume.
VSS_E_WRITERERROR_RETRYABLE 0x800423F3	The writer experienced a transient error. If the backup process is retried, the error may not reoccur.
VSS_E_WRITERERROR_TIMEOUT 0x800423F2	The writer's timeout expired between the Freeze and Thaw events.
VSS_S_ASYNC_CANCELLED 0x0004230B	The asynchronous operation has been canceled.
VSS_S_ASYNC_FINISHED 0x0004230A	The asynchronous operation has completed.
VSS_S_ASYNC_PENDING 0x00042309	The asynchronous operation is pending.
VSS_S_SOME_SNAPSHOTS_NOT_IMPORTED 0x00042321	Some shadow copies were not successfully

Error code / value	Description
	imported.

2.2.2 Constants

The following table summarizes the constants that are defined in this specification.

Constant name	Section	Description
MAX_IP_ADDRESS_LEN	2.2.2.1	Specifies the maximum length of the string representation of an IP address in structures defined in ISTM.
MAX_RESOURCE_GROUP_NAME	2.2.2.2	Specifies the maximum length of an HA cluster resource group name in structures defined in ISTM.

2.2.2.1 MAX_IP_ADDRESS_LEN

The **MAX_IP_ADDRESS_LEN** constant specifies the maximum length of the string representation of an IP address in structures defined in ISTM. The length refers to the number of Unicode characters in a string, not counting the trailing null.

```
const int MAX_IP_ADDRESS_LEN = 127;
```

2.2.2.2 MAX_RESOURCE_GROUP_NAME

The **MAX_RESOURCE_GROUP_NAME** constant specifies the maximum length of an HA cluster resource group name in structures defined in ISTM. The length refers to the number of Unicode characters in a string, not counting the trailing null.

```
const int MAX_RESOURCE_GROUP_NAME = 511;
```

2.2.3 Enumerations

The following table summarizes the enumerations that are defined in this specification.

Enumeration name	Section	Description
AuthenticationMethod	2.2.3.1	Defines methods of enumeration that can be used by a session.
DayOfWeek	2.2.3.2	Defines numeric values that correspond to the days of the week.
DeviceStatus	2.2.3.3	Defines values that describe the status of the device associated with a LUN.
HostStatus	2.2.3.4	Defines values that describe the status of a target.
IDMethod	2.2.3.5	Defines methods of initiator identification that can be used by a target.
MountType	2.2.3.6	Defines values for the type of mount and mount status of a LUN.

Enumeration name	Section	Description
SessionType	[2.2.3.7](#)	Defines values that indicate the purpose of a particular **iSCSI session**.
Type	[2.2.3.8](#)	Defines values that identify the type of file system object that is being examined.
VhdType	[2.2.3.9](#)	Defines values that identify the VHD format of a VHD file.
WTDType	[2.2.3.10](#)	Defines values that identify the type of storage from which a virtual disk has been created.

2.2.3.1 AuthenticationMethod

The **AuthenticationMethod** enumeration defines methods of authentication that can be used by a session. See [RFC3720] section 8.2 for the authentication details.

```
typedef enum _AuthenticationMethod {
    None = 0,
    Chap = 1
} AuthenticationMethod;
```

None: No authentication.

Chap: **Challenge-Handshake Authentication Protocol (CHAP)** authentication.

2.2.3.2 DayOfWeek

The **DayOfWeek** enumeration defines numeric values that correspond to the days of the week.

```
typedef enum _DayOfWeek {
    Sunday = 0,
    Monday = 1,
    Tuesday = 2,
    Wednesday = 3,
    Thursday = 4,
    Friday = 5,
    Saturday = 6
} DayOfWeek;
```

Sunday: The value that corresponds to Sunday.

Monday: The value that corresponds to Monday.

Tuesday: The value that corresponds to Tuesday.

Wednesday: The value that corresponds to Wednesday.

Thursday: The value that corresponds to Thursday.

Friday: The value that corresponds to Friday.

Saturday: The value that corresponds to Saturday.

The DayOfWeek values are used in snapshot scheduling.

2.2.3.3 DeviceStatus

The **DeviceStatus** enumeration defines values that describe the status of a LUN when it is viewed as a device from the initiator.

```
typedef enum _DeviceStatus {
    Idle = 0,
    InUse = 1,
    Reverting = 2
} DeviceStatus;
```

Idle: The LUN is not currently being used.

InUse: The LUN is being used by an initiator.

Reverting: The LUN is in the process of being rolled back.

A LUN that is not **Idle** or **InUse** SHOULD NOT participate in REPORT_LUN operations. See [SPC-3] for information concerning REPORT_LUN operations.

2.2.3.4 HostStatus

The **HostStatus** enumeration defines values that describe the status of a target.

```
typedef enum _HostStatus {
    HostStatusIdle = 0,
    HostStatusConnected = 1
} HostStatus;
```

HostStatusIdle: The target is not currently logged in.

HostStatusConnected: The target is currently logged in.

2.2.3.5 IDMethod

The **IDMethod** enumeration defines methods of initiator identification that a target can use.

```
typedef enum _IDMethod {
    IDInvalid = -1,
    IDNetBiosName = 0,
    IDDNSName = 1,
    IDIPAddress = 2,
    IDMACAddress = 3,
    IDIQN = 4,
    IDIPv6Address = 5
} IDMethod;
```

IDInvalid: Invalid identification.

IDNetBiosName: The initiator is identified by using a **NetBIOS name**.

IDDNSName: The initiator is identified by using a DNS name.

IDIPAddress: The initiator is identified by using an IP address.

IDMACAddress: The initiator is identified by using a **medium access control (MAC)** address.

IDIQN: The initiator is identified by using an iSCSI qualified name (IQN).

IDIPv6Address: The initiator is identified by using an IPv6 address.

2.2.3.6 MountType

The **MountType** enumeration defines values for the type of mount and mount status of a LUN.

```
typedef enum _MountType {
    DoNotMount = 0,
    MountSnapshot = 1,
    MountDirect = 2
} MountType;
```

DoNotMount: The LUN is dismounted.

MountSnapshot: LUN is mounted from a snapshot.

MountDirect: The LUN is locally mounted from its underlying original backing storage.

2.2.3.7 SessionType

The **SessionType** enumeration defines values that indicate the purpose of a particular iSCSI session.

```
typedef enum _SessionType {
    Invalid = 0,
    Discovery = 1,
    Normal = 2
} SessionType;
```

Invalid: The session type has not yet been specified. This value typically implies that the session is in the process of being created, or that some type of error was encountered during the creation of the session. This state SHOULD only be a transient state.

Discovery: The session was created so that the iSCSI initiator can obtain information about a particular iSCSI target. For more information, see [RFC3720] section 3.3 "iSCSI Session Types".

Normal: The session was created for the purpose of normal iSCSI data transfer between the iSCSI initiator and the iSCSI target.

2.2.3.8 Type

The **Type** enumeration defines values that identify the type of file system object that is being examined.

```
typedef enum _Type {
    Directory = 0,
    File = 1
} Type;
```

Directory: The file system object is a directory.

File: The file system object is a file.

2.2.3.9 VhdType

The **VhdType** enumeration defines values that identify the VHD format of a VHD file. For more information, see [MSDN-VHD].

```
typedef enum _VhdType {
    InvalidVhd = 0,
    FixedVhd = 1,
    DifferencingVhd = 2,
    DynamicVhd = 3
} VhdType;
```

InvalidVhd: A fixed, differencing, or dynamic VHD file.

FixedVhd: A fixed VHD file.

DifferencingVhd: A differencing VHD file.

DynamicVhd: A dynamic VHD file, sometimes called "expandable".<2>

2.2.3.10 WTDType

The **WTDType** enumeration defines values that identify the type of storage from which a virtual disk has been created.

```
typedef enum _WTDType {
    WTDTypeVolumeBased = 0,
    WTDTypeFileBased = 1,
    WTDTypeRamBased = 2
} WTDType;
```

WTDTypeVolumeBased: Reserved.

WTDTypeFileBased: The virtual disk has been created as a VHD file.

WTDTypeRamBased: The virtual disk has been created in RAM.

2.2.4 Structures

The following table summarizes the structures that are defined in this specification.

Structure name	Section	Description
CACHED_INITIATOR	2.2.4.1	Identifies an iSCSI initiator that has been discovered by the ISTM server.
CONNECTION	2.2.4.2	Contains information about an **iSCSI connection**.
DIRECTORY_DATA	2.2.4.3	Contains information about a particular file or directory.
DISK	2.2.4.4	Contains operating system information about a physical disk.
DISK2	2.2.4.5	Contains operating system information about a physical disk in an HA cluster configuration.
DRIVE_INFO	2.2.4.6	Contains information about a logical drive.
ID	2.2.4.7	Specifies the identification types for an initiator.
LICENSE	2.2.4.8	Contains implementation-defined information about the ISTM server installation.
LOGICAL_UNIT_IDENTIFIER	2.2.4.9	Specifies a storage identifier as a Fibre Channel Physical and Signaling Interface (FC-PH) identifier [ANSI-289-1996].
LUN_MAPPING	2.2.4.10	Specifies the LUN mapping on a target.
LUN_STATUS	2.2.4.11	Specifies current status information for a LUN.
MOUNT_POINT	2.2.4.12	Specifies the mount point associated with a LUN.
PORTAL	2.2.4.13	Contains information about the networking interfaces that are available for use with iSCSI traffic.
RESOURCE_GROUP	2.2.4.14	Specifies the resource group name in an HA cluster configuration.
SESSION	2.2.4.15	Contains information about an iSCSI session.
SESSION2	2.2.4.16	Contains extended information about an iSCSI session.
SNAPSHOT_SCHEDULE_SETTINGS	2.2.4.17	Specifies a schedule for performing snapshot backups of a LUN.
SNS_SERVER	2.2.4.18	Identifies an iSNS server.

Structure name	Section	Description
VOLUME	2.2.4.19	Contains information about a storage volume.
VOLUME2	2.2.4.20	Contains information about a storage volume in an HA cluster configuration.
WINTARGET_DISK_DEVICE_IDENTIFIER	2.2.4.21	Specifies a vendor-specific storage identifier.

2.2.4.1 CACHED_INITIATOR

The **CACHED_INITIATOR** structure identifies an iSCSI initiator that has been discovered by the ISTM server.

```
typedef struct _CACHED_INITIATOR {
    WCHAR szInitiatorIqn[512];
    hyper hyLastLoggedOnTime;
} CACHED_INITIATOR;
```

szInitiatorIqn: The IQN of the initiator.

hyLastLoggedOnTime: A time stamp in Coordinated Universal Time (UTC) of the last login attempt by the initiator. The value of this member SHOULD be interpreted as a **FILETIME** structure (section 2.2).

2.2.4.2 CONNECTION

The **CONNECTION** structure contains information about an iSCSI connection.

```
typedef struct _CONNECTION {
    DWORD CID;
    WCHAR InitiatorIpAddress[MAX_IP_ADDRESS_LEN + 1];
    DWORD InitiatorPort;
    WCHAR TargetIpAddress[MAX_IP_ADDRESS_LEN + 1];
    DWORD TargetPort;
    boolean HeaderDigestEnabled;
    boolean DataDigestEnabled;
} CONNECTION;
```

CID: A unique identifier for an iSCSI connection.

InitiatorIpAddress: The string representation of the IP address of the iSCSI initiator.

InitiatorPort: The connection's TCP port number on the initiator.

TargetIpAddress: The string representation of the IP address of the iSCSI target.

TargetPort: The connection's TCP port number on the target.

HeaderDigestEnabled: A Boolean value that specifies whether iSCSI messages using a particular connection contain a header digest, as described in [RFC3720] section 10.2.3.

DataDigestEnabled: A Boolean value that specifies whether iSCSI messages using a particular connection contain a data digest, as described in [RFC3720] section 10.2.3.

2.2.4.3 DIRECTORY_DATA

The **DIRECTORY_DATA** structure contains information about a particular file or directory.

```
typedef struct _DIRECTORY_DATA {
    WCHAR szFileName[512];
    WCHAR szFullPath[512];
    Type eType;
} DIRECTORY_DATA;
```

szFileName: The name of the file or directory.

szFullPath: The full path of the file or directory. The full path includes the name, as given in **szFileName**.

eType: A value from the **Type** enumeration (section 2.2.3.8) that specifies whether the object is a directory or a file.

2.2.4.4 DISK

The **DISK** structure contains operating system information about a physical disk.

```
typedef struct _DISK {
    DWORD dwIndex;
    WCHAR szWin32Path[64];
    hyper hySize;
} DISK;
```

dwIndex: The operating system index for the disk.

szWin32Path: The operating system path to the disk.<3>

hySize: The size, in bytes, of the disk.

2.2.4.5 DISK2

The **DISK2** structure contains operating system information about a physical disk in an HA cluster configuration.

```
typedef struct _DISK2 {
    DWORD dwIndex;
    WCHAR szWin32Path[64];
    hyper hySize;
    WCHAR szResourceGroup[MAX_RESOURCE_GROUP_NAME + 1];
} DISK2;
```

dwIndex: The operating system index for the disk.

szWin32Path: The operating system path to the disk.<4>

hySize: The size, in bytes, of the disk.

szResourceGroup: The name of the owning resource group. If the disk is not part of a resource group, the character array MUST be empty; that is, it MUST contain zeros.

2.2.4.6 DRIVE_INFO

The **DRIVE_INFO** structure contains information about a logical drive.

```
typedef struct _DRIVE_INFO {
    WCHAR szDrive[20];
    hyper hySize;
    hyper hyFreeSpace;
    boolean bSupportSparseFiles;
} DRIVE_INFO;
```

szDrive: The mount point of the logical drive; for example, "C:\".

hySize: The total size, in bytes, of the logical drive.

hyFreeSpace: The amount of free space, in bytes, that is available on the logical drive.

bSupportSparseFiles: A Boolean value that specifies whether the logical drive supports **sparse files**.

2.2.4.7 ID

The **ID** data structure specifies the identification types of an initiator towards a target.

```
typedef struct _ID {
    IDMethod eMethod;
    WCHAR szValue[512];
} ID;
```

eMethod: The identification method type, which is specified by an **IDMethod** value (section 2.2.3.5).

szValue: The initiator identifier. The format of the identifier depends on the method of identification, and it follows the naming convention and restrictions of the methods.

The following table describes the initiator identifier for various identification methods.

ID method	Initiator identifier
IDNetBIOSNames	The string is the NetBIOS name of the initiator. For naming restrictions in length and character, see [RFC1001]. The **IDNetBIOSNames** value SHOULD NOT be used.

ID method	Initiator identifier
IDDNSNames	The string is the **fully qualified domain name (FQDN)** of the initiator. For examples and definition of a valid DNS name, see [RFC1123].
IDMACaddress	The string is the representation of the medium access control (MAC) address, as specified in [IEEE802.1X]**,** represented in MAC-48 form, dash-separated.
IDIQN	The string is the IQN name of the initiator [RFC3720]**.**
IDIPAddresses	The string is the **IPv4** address in dotted-decimal notation.
IDIPv6Addresses	The string is the IPv6 address in colon-hexadecimal notation.

2.2.4.8 LICENSE

The **LICENSE** structure contains implementation-defined information about the ISTM server installation.<5>

```
typedef struct _LICENSE {
    boolean bDemo;
    unsigned int nNumDemoDaysLeft;
    unsigned int nNumSeats;
    unsigned int nFeatures;
    WCHAR LicenseKeys[10][64];
} LICENSE;
```

bDemo: A Boolean value that specifies whether the ISTM protocol is installed for temporary demonstration purposes.

nNumDemoDaysLeft: An unsigned integer that specifies the number of days remaining that the ISTM installation is allowed to be used for demonstration purposes. This value is meaningful only if **bDemo** is true.

nNumSeats: An unsigned integer that specifies the number of concurrent users of the ISTM installation.

nFeatures: An unsigned integer that specifies the iSCSI Software Target management features that have been activated in the ISTM installation.

LicenseKeys: A 2-dimensional array of strings that specify the license keys for the ISTM installation.

2.2.4.9 LOGICAL_UNIT_IDENTIFIER

The **LOGICAL_UNIT_IDENTIFIER** structure specifies a storage identifier as a Fibre Channel Physical and Signaling Interface (FC-PH) identifier. Those identifiers consist of 4 bits of Network Address Authority followed by 124 bits of company and vendor-specific information. See [RFC3980] and [ANSI-289-1996] for further information.

```
typedef struct _LOGICAL_UNIT_IDENTIFIER {
    BYTE NaaType_CompanyIdHigh;
    BYTE CompanyIdMid[2];
```

```
        BYTE CompanyIdLow_VsidHigh;
        DWORD VsidLow;
        hyper VsidExtension;
    } LOGICAL_UNIT_IDENTIFIER;
```

NaaType_CompanyIdHigh: The Network Address Authority type of the identifier in the 4 most-significant bits of this value, and the 4 most-significant bits of the company identifier in the 4 least-significant bits of this value.

CompanyIdMid: The 16 next most-significant bits of the company identifier.

CompanyIdLow_VsidHigh: The 4 least-significant bits of the company identifier in the 4 most-significant bits of this value, and the 4 most-significant bits of the vendor-specific identifier extension in the 4 least-significant bits of this value.

VsidLow: The 16 next most-significant bits of the vendor-specific identifier extension.

VsidExtension: The 64 least-significant bits of the vendor-specific identifier extension. The value of this member SHOULD be interpreted as an unsigned value.

2.2.4.10 LUN_MAPPING

The **LUN_MAPPING** structure specifies the LUN mapping on a target.

```
    typedef struct _LUN_MAPPING {
        DWORD dwLun;
        DWORD dwWTD;
    } LUN_MAPPING;
```

dwLun: The LUN number, as represented to the initiator [SAM-3].

dwWTD: The disk index, as represented and maintained by the LUN manager.

2.2.4.11 LUN_STATUS

The **LUN_STATUS** structure specifies current status information for a LUN.

```
    typedef struct _LUN_STATUS {
        DWORD dwLun;
        DeviceStatus eStatus;
        DWORD dwProgressPercent;
        DWORD dwCompletionError;
    } LUN_STATUS;
```

dwLun: The disk index, as represented and maintained by the LUN manager.

eStatus: A **DeviceStatus** value (section 2.2.3.3) that specifies the status of the LUN when viewed as a device from the initiator.

dwProgressPercent: The percentage of rollback progress of the LUN. The value MUST be in the range of 0 to 100, inclusive.

dwCompletionError: The completion code for the rollback process.

2.2.4.12 MOUNT_POINT

The **MOUNT_POINT** structure specifies the mount point associated with a LUN.

```
typedef struct _MOUNT_POINT {
    WCHAR szPath[512];
} MOUNT_POINT;
```

szPath: The path of the mount point associated with a LUN.

2.2.4.13 PORTAL

The **PORTAL** structure contains information about the networking interfaces that are available for use with iSCSI traffic.

```
typedef struct _PORTAL {
    GUID Guid;
    WCHAR szIpAddress[MAX_IP_ADDRESS_LEN + 1];
    DWORD dwPort;
    BOOL bListen;
    BOOL bSecure;
    BOOL bValid;
} PORTAL;
```

Guid: A **GUID** (section 2.2) that is created when portal information is collected by the iSCSI Software Target.

szIpAddress: The string representation of an IP address assigned to a network interface.

dwPort: The TCP port number that is being used for iSCSI traffic through the network interface.

bListen: A Boolean value that specifies whether the iSCSI Software Target is listening on the network interface.

bSecure: A Boolean value that specifies whether the portal is secured by **Internet Protocol security (IPsec)**.<6>

bValid: A Boolean value that specifies whether the network interface is viable for iSCSI traffic.

2.2.4.14 RESOURCE_GROUP

The **RESOURCE_GROUP** structure specifies the resource group name in an HA cluster configuration.

```
typedef struct _RESOURCE_GROUP {
    WCHAR szName[MAX_RESOURCE_GROUP_NAME + 1];
} RESOURCE_GROUP;
```

szName: The resource group name.

2.2.4.15 SESSION

The **SESSION** structure contains information about an iSCSI session.

```
typedef struct _SESSION {
    WCHAR InitiatorIQN[256];
    WCHAR HostTargetIQN[256];
} SESSION;
```

InitiatorIQN: The IQN of the iSCSI initiator of a particular iSCSI session.

HostTargetIQN: The IQN of the iSCSI target of a particular iSCSI session.

2.2.4.16 SESSION2

The **SESSION2** structure contains extended information about an iSCSI session.

```
typedef struct _SESSION2 {
    WCHAR InitiatorIQN[256];
    WCHAR HostTargetIQN[256];
    WCHAR HostName[512];
    WORD TSIH;
    DWORDLONG ISID;
    SessionType eType;
    AuthenticationMethod eAuthMethod;
} SESSION2;
```

InitiatorIQN: The IQN of the iSCSI initiator of a particular iSCSI session.

HostTargetIQN: The IQN of the iSCSI target of a particular iSCSI session.

HostName: The name of the iSCSI target that was specified when the target was created.

TSIH: The target-session-identifying handle (TSIH) as described in [RFC3720].

ISID: The initiator session identifier (ISID) as described in [RFC3720].

eType: A **SessionType** value (section 2.2.3.7) that specifies the session type of a particular iSCSI session, as specified in [RFC3720] section 3.3.

eAuthMethod: An **AuthenticationMethod** value (section 2.2.3.1) that specifies the authentication method of a particular iSCSI session, as specified in [RFC3720] section 11.

2.2.4.17 SNAPSHOT_SCHEDULE_SETTINGS

The **SNAPSHOT_SCHEDULE_SETTINGS** structure specifies a schedule for performing snapshot backups of a LUN.

```
typedef struct _SNAPSHOT_SCHEDULE_SETTINGS {
    boolean bEveryday;
    DayOfWeek eDay;
    hyper hyStartTime;
} SNAPSHOT_SCHEDULE_SETTINGS;
```

bEveryday: A Boolean value that specifies whether to perform the snapshot backup every day.

eDay: A **DayOfWeek** value (section 2.2.3.2) that specifies the day of the week to perform the snapshot backup.

hyStartTime: The time of day to start the snapshot backup in Coordinated Universal Time (UTC). The value of this member SHOULD be interpreted as a **FILETIME** structure (section 2.2). A value of zero indicates that a snapshot backup is not scheduled.

2.2.4.18 SNS_SERVER

The **SNS_SERVER** structure identifies an iSNS server.

```
typedef struct _SNS_SERVER {
    WCHAR szName[512];
    BOOL bDHCPDiscovered;
} SNS_SERVER;
```

szName: A string that contains the DNS name or IP address of the iSNS server.

bDHCPDiscovered: A Boolean value that specifies whether the iSNS server was discovered on the network using the **Dynamic Host Configuration Protocol (DHCP)**.

2.2.4.19 VOLUME

The **VOLUME** structure contains information about a storage volume.

```
typedef struct _VOLUME {
    DWORD dwDiskIndex;
    WCHAR szGuid[64];
    WCHAR szMountPoint[512];
    WCHAR szLabel[64];
    WCHAR szFileSystem[64];
    boolean bSystemVolume;
    boolean bPageFile;
    hyper hyOffset;
    hyper hySize;
    hyper hyTotalVolSize;
    BOOL bValidWTVolume;
    BOOL bIsContainedInExtendedPartition;
    BOOL bIsOnDynamicDisk;
    BOOL bUnallocated;
} VOLUME;
```

dwDiskIndex: The operating system's index for the disk that hosts the volume.

szGuid: The **GUID** (section 2.2) of the volume.

szMountPoint: The mount point of the volume.

szLabel: The name of the volume.

szFileSystem: The name of the file system that is deployed on the volume; for example, "NTFS".

bSystemVolume: A Boolean value that specifies whether the volume is a **system volume** or boot device.

bPageFile: A Boolean value that specifies whether the volume contains a **page file**.

hyOffset: The offset of the volume from the beginning of the hosting disk, in bytes.

hySize: The size of the volume, in bytes, that is available for file system usage.

hyTotalVolSize: The total size of the volume, in bytes.

bValidWTVolume: A Boolean value that specifies whether this volume can be used by the iSCSI Software Target.

bIsContainedInExtendedPartition: A Boolean value that specifies whether the volume is within an extended partition.

bIsOnDynamicDisk: A Boolean value that specifies whether the volume resides on a **dynamic disk**.

bUnallocated: A Boolean value that specifies whether the volume represents unallocated space.

2.2.4.20 VOLUME2

The **VOLUME2** structure contains information about a storage volume in an HA cluster configuration.

```
typedef struct _VOLUME2 {
    DWORD dwDiskIndex;
    WCHAR szGuid[64];
    WCHAR szMountPoint[512];
    WCHAR szLabel[64];
    WCHAR szFileSystem[64];
    boolean bSystemVolume;
    boolean bPageFile;
    hyper hyOffset;
    hyper hySize;
    hyper hyTotalVolSize;
    hyper hyFreeSpace;
    BOOL bValidWTVolume;
    BOOL bIsContainedInExtendedPartition;
```

```
      BOOL bIsOnDynamicDisk;
      BOOL bUnallocated;
      BOOL bSupportSparseFiles;
      BOOL bIsQuorum;
      WCHAR szResourceGroup[MAX_RESOURCE_GROUP_NAME + 1];
      hyper MaxFileSize;
  } VOLUME2;
```

dwDiskIndex: The operating system's index for the disk that hosts the volume.

szGuid: The **GUID** (section 2.2) of the volume.

szMountPoint: The mount point of the volume.

szLabel: The name of the volume.

szFileSystem: The name of the file system that is deployed on the volume; for example "NTFS".

bSystemVolume: A Boolean value that specifies whether the volume is a system volume or boot device.

bPageFile: A Boolean value that specifies whether the volume contains a page file.

hyOffset: The offset of the volume from the beginning of the hosting disk, in bytes.

hySize: The size of the volume, in bytes, that is available for file system usage.

hyTotalVolSize: The total size of the volume, in bytes.

hyFreeSpace: The total amount of free space, in bytes, that is left on the volume.

bValidWTVolume: A Boolean value that specifies whether this volume can be used by the iSCSI Software Target.

bIsContainedInExtendedPartition: A Boolean value that specifies whether the volume is within an extended partition.

bIsOnDynamicDisk: A Boolean value that specifies whether the volume resides on a dynamic disk.

bUnallocated: A Boolean value that specifies whether the volume represents unallocated space.

bSupportSparseFiles: A Boolean value that specifies whether the volume supports sparse files.

bIsQuorum: A Boolean value that specifies whether the volume is used in some way by the consensus distributed-algorithm run by the nodes in an HA cluster.

szResourceGroup: The name of the owning resource group. If the volume is not part of a resource group, the array MUST be empty; that is, contain zeros.

MaxFileSize: The maximum file size allowed on the volume by the file system.

2.2.4.21 WINTARGET_DISK_DEVICE_IDENTIFIER

The WINTARGET_DISK_DEVICE_IDENTIFIER specifies a vendor-specific storage identifier.

```
typedef struct _WINTARGET_DISK_DEVICE_IDENTIFIER {
    BYTE szSignature[10];
    BYTE szServerName[150];
    DWORD dwLun;
    GUID SnapshotId;
} WINTARGET_DISK_DEVICE_IDENTIFIER;
```

szSignature: A null-terminated string that is initialized with the well-known content "sapporo21", which identifies this structure.

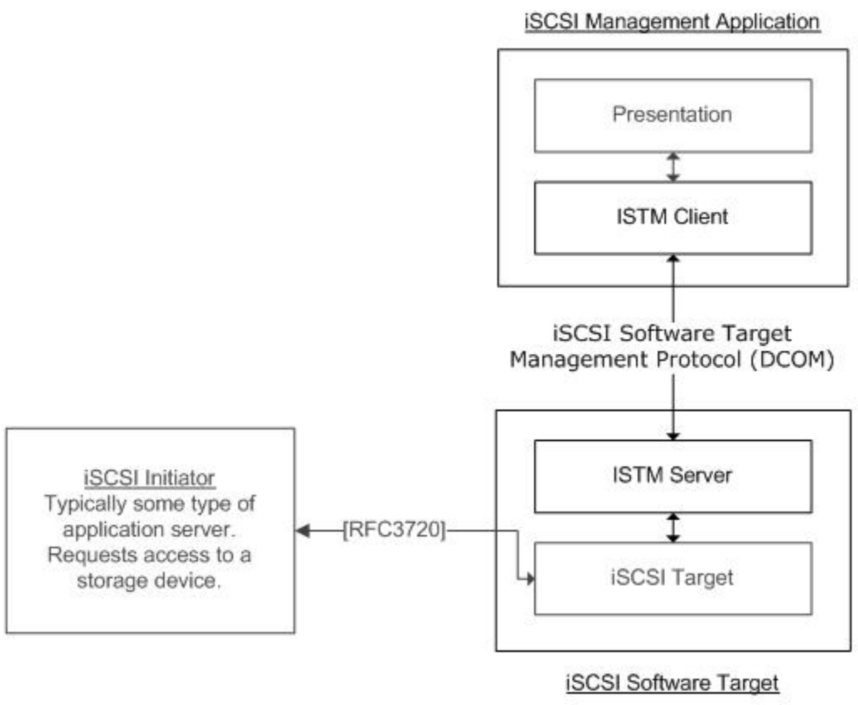
szServerName: The fully qualified domain name (FQDN) of the iSCSI target server that is supplying this information.

dwLun: A disk index, as known and maintained by the LUN manager.

SnapshotId: A **GUID** (section 2.2) that uniquely identifies the snapshot from which originated the information in the present instance of the structure.

3 Protocol Details

The following figure is similar to the figure in section 1.3 that shows a high-level view of an iSCSI environment, but this version is more detailed, showing the ISTM server and ISTM client components of the ISTM protocol.



iSCSI Management Application
Presentation
ISTM Client
iSCSI Software Target Management Protocol (DCOM)
ISTM Server
iSCSI Target
iSCSI Software Target
iSCSI Initiator
Typically some type of application server. Requests access to a storage device.
[RFC3720]

Figure 8: High-level view of an iSCSI environment showing ISTM server and client

Section 3.1 of this document specifies the details of the server role that are common to this protocol, and section 3.2 specifies the common details of the client role. Note that the server and client roles do not always correspond to the ISTM server and ISTM client components shown above. The ISTM server usually—but not always—acts in the server role, and the ISTM client usually—but not always—acts in the client role.

One of the functions of the feature management subsystem is to support the notification of changes in status in the ISTM server. The ISTM client can register with the ISTM server to receive status notifications, and when those notifications are made, the ISTM server acts in the client role and the ISTM client acts in the server role.

The ISTM client can register with the ISTM server to receive notification of events including the following:

- A new LUN is created.
- The LUN has been assigned to an iSCSI target.
- An iSCSI target stopped using a LUN.
- Snapshot restore progress.

3.1 Common Server Details

The following diagram shows the major relationship between actors in the ISTM server. These actors are managed by the subsystems described in section [1.3](#).

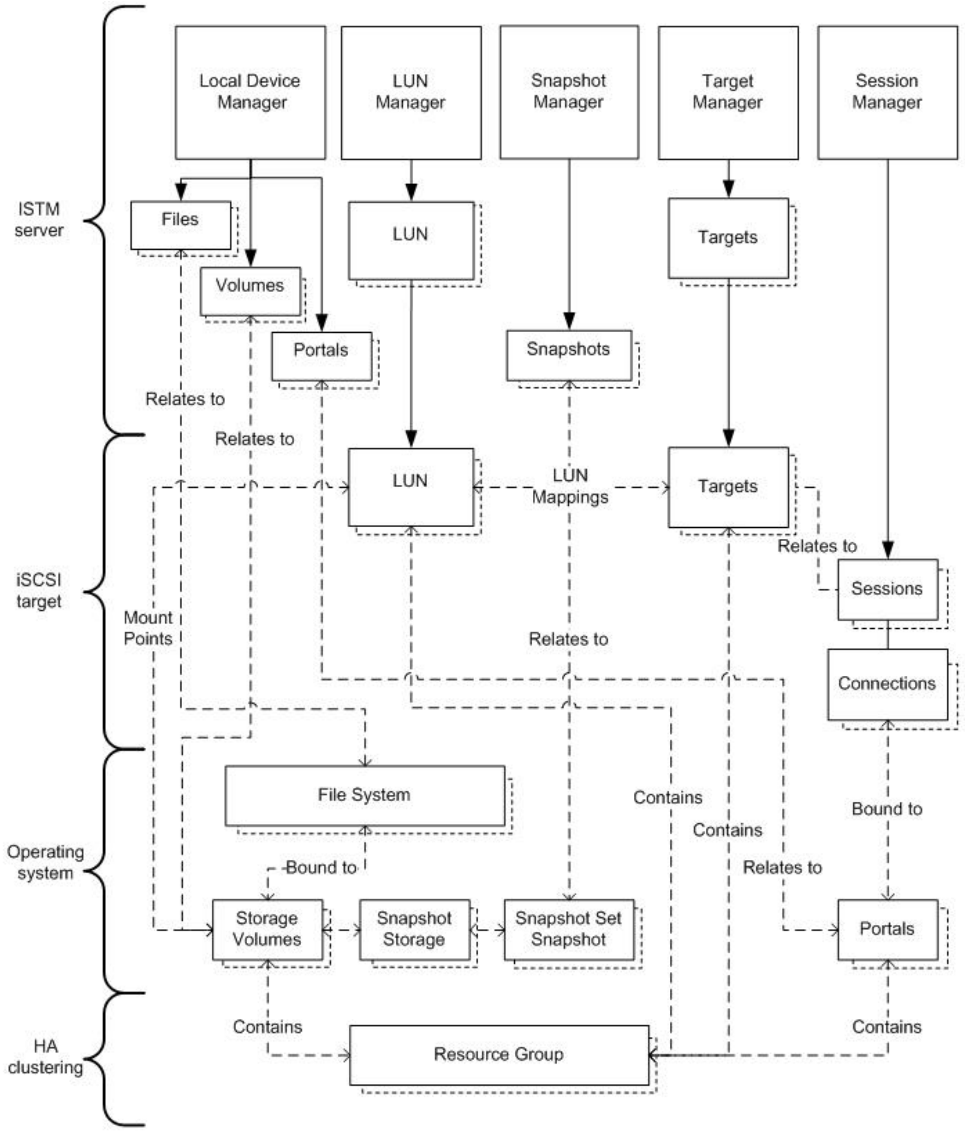


Figure 9: Subsystem relationships in the ISTM server

The ISTM server manages an iSCSI target implementation that is built upon generally available operating system facilities and, optionally, upon a generic HA clustering infrastructure.

The HA clustering infrastructure maintains administrative groups of resources such as storage volumes or disks, network adapters, and DNS names across a set of computers, called nodes of the cluster [MS-CMRP]. Each resource group is identified by a name, unique in the domain of the HA clustering infrastructure.

Resource groups include resources such as storage volumes and IP addresses assigned to network adapters, and they have node owners. The storage volumes can host file systems, and the file systems can host VHD files that can server as the underlying implementation of the LUNs exposed by the LUN manager in the ISTM server. The network adapters can have IP Addresses. Those IP Addresses can be associated with iSCSI portals, to which an iSCSI initiator opens a connection bound to a session. The connections and sessions are exposed by the session manager in the ISTM server.

An ISTM client manages iSCSI targets through the target management subsystem. By request from the ISTM client, the target manager creates target objects, and the ISTM server uses the target manager to maintain the collection of targets. Using the methods exposed by the **IEnumHost** interface (section 3.8), an ISTM client can enumerate the targets in an ISTM server or all the targets in a particular resource group in an HA cluster environment.

An **IHost** interface pointer is an abstraction of a target. It is created by the target manager on demand from an ISTM client. The client can add a LUN to a specific target or remove a particular LUN or all LUN objects from a particular target.

Through the LUN manager, the ISTM server maintains a collection of LUN objects associated with each target. An ISTM client can enumerate LUN objects associated with a specific target, as well as the association of those LUNs with the iSCSI LUN number [SAM-3] by using the methods exposed by the **IEnumWTDiskLunMapping** interface (section 3.20).

Through the local device and target managers, the ISTM server manages the **iSCSI initiator portals** that service the storage requests for a target, as well as the storage volume suitable for hosting a LUN. An ISTM client can enumerate the portals for a specific target using the methods exposed by the **IEnumPortal** interface (section 3.11). The ISTM server maintains portal information for each portal as specified by the **PORTAL** structure (section 2.2.4.13). An ISTM client can enumerate the storage volumes that are usable by the ISTM server by using the methods exposed by the **IEnumVolume2** interface.

Through the session manager, the ISTM server maintains the collection of the sessions that are associated with each target. An ISTM client can enumerate the sessions associated with a specific target using the methods exposed by the **IEnumSessions2** interface.

The LUN manager object is an abstraction for managing LUN objects. The LUN objects are referred to in the **Interface Definition Language (IDL)** as **IWTDisk** interface pointers. The LUN manager is a singleton object that is created at ISTM server startup. A client, with the help of the exposed LUN manager interface, can create new LUN objects and delete LUN objects.

The LUN manager exposes the **IEnumWTDisk** interface to enumerate all LUN objects. In an HA cluster environment, this interface also enumerates LUN resources that are not in the online state. The client can then obtain the **ResourceState** property of each LUN to determine the state of the resource.

3.1.1 Abstract Data Model

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

The following data object is common to the subsystems of the ISTM server:

Iterator: A cursor mechanism used to iterate through a collection of enumerated objects of a certain type on the ISTM server when the **Next** method of an enumeration interface is called by an ISTM client.

The following subsections present the abstract data models for the functional subsystems introduced in section 1.3.

3.1.1.1 Feature Management Data Model

The following diagram shows some of the relationships between actors in the ISTM server for the feature management data model.

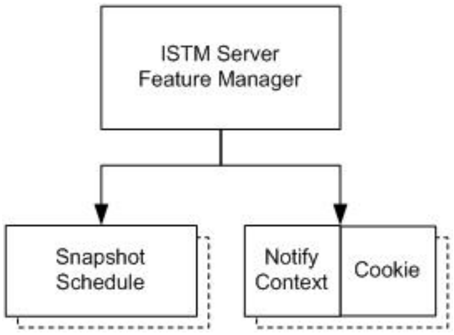


Figure 10: ISTM server feature manager

A data model is maintained by the feature management subsystem to support general feature management. It includes the following persisted data:

Guid: A **GUID** (section 2.2) that identifies the ISTM server service.

Name: A string that uniquely identifies the iSCSI Software Target. This name is for informative purposes only; the domain of the name's uniqueness is not specified or mandated by this protocol.

ScheduleList: A write-only collection of work-items, to be executed at a specified point in time. Each element of this collection is a **Schedule** data object.

Schedule: A work-item description that contains a snapshot schedule. The snapshot schedule specifies parameters for performing the backup of data, including the following information:

- Whether the backup occurs every day.
- The day of the week for the backup.
- The time of day to start the backup.

A data model is maintained by the feature management subsystem to support status notifications. It is described in the **IStatusNotify** client abstract data model (section 3.42.1).

3.1.1.2 Session Management Data Model

The following diagram shows some of the relationships between actors in the ISTM server for the session management data model.

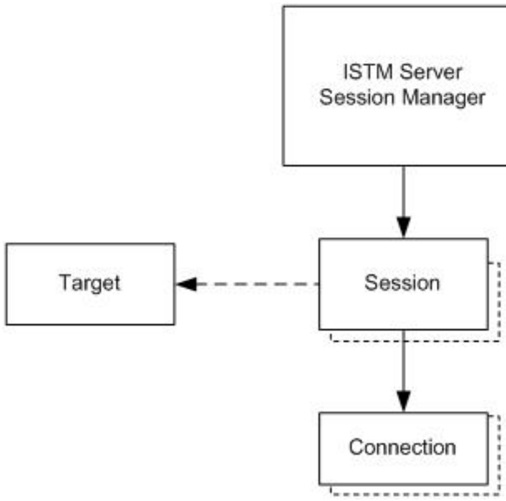


Figure 11: ISTM server session manager

The session manager maintains a collection of sessions. Each session is logically related with one iSCSI target. Each session maintains a collection of connections that are related to that session. The following subsections describe the session and connection data.

3.1.1.2.1 iSCSI Session Data

An ISTM server that implements session data interfaces maintains the following session data either directly or, more likely, by sharing information with an underlying iSCSI target implementation.

InitiatorName: The IQN of the initiator that was supplied as part of session establishment.

TargetName: The IQN of the target as it was specified by the initiator as part of session establishment.

SessionType: For each iSCSI session, a variable that stores the session type, as described in [RFC3720] section 3.3.

AuthenticationMethod: For each iSCSI session, a variable that stores the authentication method of the session, as described in [RFC3720] section 11.

TargetSessionID: The target session identifying handle (TSIH) as described in [RFC3720].

InitiatorSessionID: The initiator session identifier (ISID) as described in [RFC3720].

3.1.1.2.2 iSCSI Connection Data

An ISTM server that implements session data interfaces maintains the following connection data either directly or, more likely, by sharing information with an underlying iSCSI target implementation.

InitiatorIpAddress: The IP address of the iSCSI initiator for a particular iSCSI connection.

InitiatorPort: The port number of the iSCSI initiator for a particular iSCSI connection.

TargetIpAddress: The IP address of the iSCSI target for a particular iSCSI connection.

TargetPort: The port number of the iSCSI target for a particular iSCSI connection.

HeaderDigestEnabled: A variable that specifies whether iSCSI messages using a particular connection contain a header digest, as described in [RFC3720] section 10.2.3.

DataDigestEnabled: A variable that specifies whether iSCSI messages using a particular connection contain a data digest, as described in [RFC3720] section 10.2.3.

3.1.1.3 Local Device Management Data Model

The following diagram shows some of the relationships between actors in the ISTM server for the local device management data model.

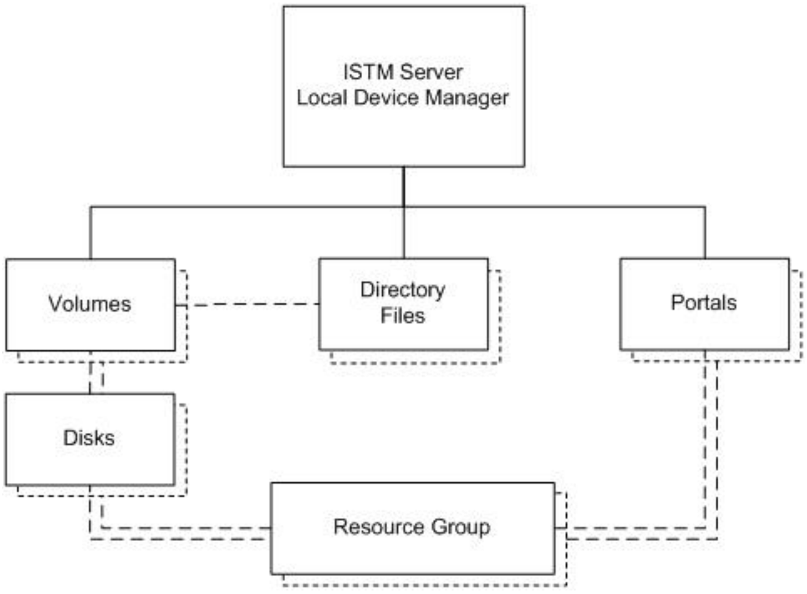


Figure 12: ISTM server local device manager

An ISTM server that implements the local device manager interfaces maintains collections of disks, volumes, files, directories (organized in a hierarchical file system), and network communication portals. In an implementation of an ISTM server, some of those entities might have counterparts in native operating system constructs and concepts.

The following subsections describe the data objects managed by this subsystem.

3.1.1.3.1 Disk Data

The ISTM server maintains information about disks on the iSCSI Software Target computer, and it can perform certain operations on them. A disk is a unique and non-divisible physical entity that represents a persistent storage device.

A disk has a unique physical identity, such as a bus address on the particular technology that supports the disk, and a specific mechanism by which it is exposed and addressed by the underlying operating system. Each disk has an associated index number, which is maintained by the operating system, and which abstracts the different bus access and bus addressing technologies.

A disk can be divided into smaller logical entities, called partitions or logical disks. Each partition can host a volume or contain disk metadata. However, because disks are physically indivisible and are uniformly accessed from a unique, hardware-specific access point, they can only be moved or owned as whole entities.

Different disks or partitions on a disk can be pooled to form volumes according to implementation-specific policies. Dynamic disks offer greater flexibility for volume management because they use a database to track information about **dynamic volumes** on the disk and about other dynamic disks in the computer. Dynamic volumes can encompass multiple physical disks. Types of dynamic volumes include spanned, striped and mirrored.

In HA cluster configurations, physical disks are also cluster resources, and they can be assigned to resource groups and owned by computer nodes.

Information about physical disks is maintained by the underlying operating system and can be used to support the management of disks on the iSCSI Software Target computer. A collection of objects that are represented by the **DISK** structure (section 2.2.4.4) can be used for disk enumeration by the **IEnumDisk** interface (section 3.6). A collection of objects that are represented by the **DISK2** structure (section 2.2.4.5) can be used for disk enumeration by the **IEnumDisk2** interface (section 3.7).

3.1.1.3.2 Volume Data

The iSCSI Software Target computer can directly maintain or can surface or expose a number of volumes. A volume is the minimal uniquely identifiable storage entity that can host a file system. The ISTM server can enumerate volumes that exist in the iSCSI Software Target computer that are compatible to the operations that the iSCSI Software Target supports. It can also return information for a particular volume.

Information about volumes that are available for the creation of LUNs is maintained by the underlying operating system and can be used to support the management of volumes on the iSCSI Software Target computer. A collection of objects that are represented by the **VOLUME** structure (section 2.2.4.19) can be used for volume enumeration by the **IEnumVolume** interface (section 3.17). A collection of objects that are represented by the **VOLUME2** structure (section 2.2.4.20) can be used for volume enumeration by the **IEnumVolume2** interface (section 3.18)

Additional abstract data is maintained by the local device management subsystem to support the management of volumes on the iSCSI Software Target computer. This data model includes the following persisted data:

Note Some of the data elements described below might not be available or meaningful to a particular volume, because they might depend on the file system that is hosted on the volume, on the volume metadata, or on other factors that are outside of the control of the ISTM server. Each description that is provided below is applicable if the underlying operating system supports the corresponding element.

DiskIndex: An identifier that correlates the volume with the phisical disk where it resides, if any exists or is applicable. This index is maintained by the operating system implementation.

VolumeGuid: The unique identifier, in the form of a **GUID**, of a volume in the device-manager scope.

VolumeMountPoint: The set of paths from which the volume content is accessible from the file system.

VolumeLabel: The name of the volume.

FileSystem: The type of file system that created the volume.

SystemVolumeFlag: Specifies whether the volume is a system volume. In certain implementations this flag might indicate that the operating system installation resides on that volume.

PageFileFlag: Specifies whether the volume has a page file. In certain implementations this flag might indicate that a swap-file or a paging file resides on that volume.

VolumeOffset: The offset of the volume from the start of the disk.

VolumeSize: The size of the volume available for file system usage.

TotalVolumeSize: The total size of the volume.

VolumeFreeSpace: The total free space that is left on the volume. The free space is managed by the file system that is hosted on the volume.

IsContainedInExtendedPartitionFlag: Specifies whether the partition is within an extended partition. This flag is used only when the volume information represents an unallocated space.

IsOnDynamicDiskFlag: Specifies whether the volume resides on a dynamic disk. This flag is used only when the volume information represents an unallocated space.

UnallocatedFlag: Specifies whether the volume represents unallocated space.

SupportSparseFilesFlag: Specifies whether the drive supports sparse files.

IsQuorumFlag: In an HA cluster configuration, specifies whether the volume is used in some way by the consensus distributed-algorithm run by the nodes in the cluster.

VolumeResourceGroup: In an HA cluster configuration, the name of the resource group that owns the volume.

VolumeMaxFileSize: The maximum file size allowed on the volume by the file system.

Note Some or all of the data captured by these ADM elements might be available from the operating system of the iSCSI Software Target computer.

3.1.1.3.3 File Data

The ISTM server maintains information about each LUN that is created to represent a virtual disk to the iSCSI initiator. The ISTM server maintains relationships between the LUN and their physical representations through a file system path to a VHD file. The local device manager maintains or exposes from the underlying operating system the following data about files.

SparseFile: A flag that specifies whether the file is a sparse file.

DevicePath : The path to a file in the domain of the file system.

FileSystem: The type of file system where the **DevicePath** resides.

AllocatedSize: The size of the file as maintained by the file system.

3.1.1.3.4 Directory Data

The ISTM server can return information about the directories that exist on a file system.

DirName: The name of the directory, valid within a specific context.

DirFullPath: The full path of the directory.

PathType: Specifies whether the information returned by the ISTM server is a directory or a file.

Note Some or all of the data captured by these ADM elements might be available from the operating system of the iSCSI Software Target computer.

3.1.1.3.5 Portal Data

The ISTM server maintains information about networking interfaces that are available in HA cluster and non-HA cluster configurations and their configuration as iSCSI initiator portals [RFC3720].

Information about portals is maintained by the underlying operating system and can be used to support the management of portals on the iSCSI Software Target computer. A collection of objects that are represented by the **PORTAL** structure (section 2.2.4.13) can be used for portal enumeration by the **IEnumPortal** interface (section 3.11). The scope of the set of portals represented in such a collection varies according to the context of the enumeration.

Additional abstract data is maintained by the local device management subsystem to support the management of portals on the iSCSI Software Target computer. This data model includes the following persisted data:

PortalGuid: The unique identifier for the portal, in the form of a GUID.

ListenFlag: A flag that indicates whether the iSCSI Software Target is listening on the associated TCP port.

PortalResourceGroup: The name of the owning resource group.

3.1.1.3.6 Processor Data

The ISTM server can return information about the processors that exist on the iSCSI Software Target computer that the iSCSI Software Target software can run on.

ProcessAffinityMask: A bit mask that specifies what processors that are available.

Note The data captured by this ADM element might be available from the operating system of the iSCSI Software Target computer.

3.1.1.3.7 Resource Group Data

In an HA cluster configuration, the ISTM server exposes relationship information between the resource groups that are configured by the cluster environment and other managed objects such as volumes, files that reside on a volume, and portals.

ResourceGroupName: The name of the resource group.

ResourceGroupGuid: The GUID of the resource group.

GroupVolumes: A collection of the volumes that are owned by the resource group. Device paths to VHD files are naturally mapped to a group by virtue of residing on a given volume.

GroupPortals: A collection of the portals that are owned by the resource group.

3.1.1.4 LUN Management Data Model

The following diagram shows some of the relationships between actors in the ISTM server for the LUN management data model.

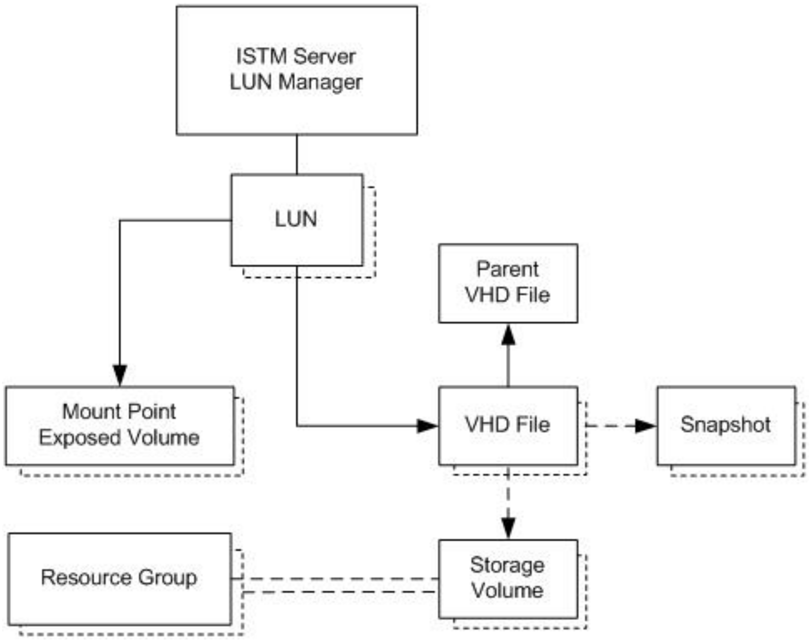


Figure 13: ISTM server LUN manager

A LUN object can be backed by several logical and physical storage implementations, including file-based or RAM-based implementations.

If a LUN is file based:

- The file can be related to point-in-time backup version of the same file.
- It might have relationships with other files such as in the differencing VHD case.
- It has a relationship with the volume that is hosting the file system, and with a volume that is dedicated to backup operations.

In an HA cluster configuration, storage volumes, which were introduced in section 3.1, have relationships with resource groups.

A LUN can be surfaced as a volume to the computer running the iSCSI Software Target.

The ISTM server maintains the collection of mount points for an exposed volume associated with a LUN. They can be enumerated with an iterator exposed through the **IEnumLMMountPoint** interface (section 3.10). The information enumerated with each mount point is specified by a **MOUNT_POINT** structure (section 2.2.4.12).

WTDiskList: A collection of **IWTDisk** interface (section 3.43) objects for all virtual disks on the iSCSI Software Target. Each IWTDisk object defines the following virtual disk properties:

Note In an HA cluster configuration, offline virtual disk cluster resources SHOULD NOT be present in the **WTDiskList** collection. Specifically, only virtual disks with a **ResourceState** data element value of **ClusterResourceOnline** ([MS-CMRP] section 3.1.4.1.13) SHOULD be present.

Guid: The unique identifier for the virtual disk, in the form of a GUID.

SerialNumber: The serial number for the virtual disk.

WTDIndex: The index of the LUN within the collection maintained by the ISTM server.

Type: A value that specifies whether the virtual disk object is file-based, volume-based or RAM-based, as defined by the **WTDType** enumeration (section 2.2.3.10).

Flags: The mapping flag, which SHOULD be one or more of the following bit settings:

Name/Value	Description
WTDFlagEnableLunMapping 1	Mapping the LUN to a VHD disk is allowed.
WTDFlagShadowLun 2	The LUN is mapped to a **shadow copy** of a VHD disk.

Status: The status of the device, as specified by the **DeviceStatus** enumeration (section 2.2.3.3).

Description: A user-friendly description of the disk object, as specified by the client.

DevicePath: The path to the disk in the naming domain of its type. For a file-based LUN, it is the file system path of the VHD file.

Size: The size of the disk in bytes.

SnapshotStoragePath: The path to the volume or storage facility that is associated with this LUN for backup operations.

SnapshotStorageSize: The size, in megabytes, of the space devoted to backup operations.

SnapshotStorageUsedSize: The size, in megabytes, of the storage that is currently being used for backup at the storage path.

LMType: The local mount type, as specified by the **MountType** enumeration (section 2.2.3.6).

LMTimeStamp: The local mount time stamp.

LMMountPoint: The local mount point.

LMCurrMountDeviceId: The **plug and play identifier** of a current locally mounted LUN.

LMSnapshotId: The local mount snapshot identifier.

Enable: A Boolean value that specifies whether the resource is enabled or disabled.

ResourceName: In an HA cluster configuration, the resource name of this LUN. The resource name is an alias for the resource in the naming domain of the cluster management.

ResourceGroupName: The resource group name with which the LUN is associated.

VhdType: A value that specifies the type for disk format, as specified in the **VhdType** enumeration (section 2.2.3.9):

Fixed

Differencing

Dynamic format.

AllocatedSize: The current allocated size of the VHD image. For a fixed-format VHD, this is the same as the **Size** property. For a differencing VHD, this is the current allocated size on the disk.

CacheParent: A Boolean value that specifies whether the parent VHD SHOULD be opened with file system caching enabled. This flag is active the next time the VHD is loaded. This property is only applicable to a differencing VHD.

ResourceState: The cluster state of the virtual disk resource, which SHOULD be one of the values specified in [MS-CMRP] section 3.1.4.1.13.

LunStatus: An object that contains the following status information for the LUN:

The LUN identifier.

The current device status, as specified in the **DeviceStatus** enumeration (section 2.2.3.3):

- Idle: When the LUN is not currently associated with a physical device or a file.
- In use: When the LUN is associated with a physical device or a file, or is use by an initiator.
- Reverting: When the device is in the process of a backup operation.

The completion percentage of a backup operation.

The completion code returned from a backup operation.

3.1.1.5 Snapshot Management Data Model

The following diagram shows some of the relationships between actors in the ISTM server for the snapshot management data model.

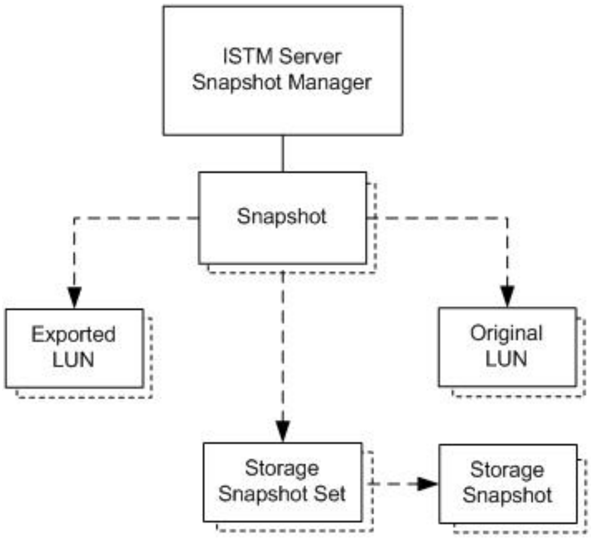


Figure 14: ISTM server snapshot manager

A data model is maintained by the snapshot management subsystem to support the management of all snapshots of the LUNs managed by the LUN manager.

The snapshot manager assumes the existence of an underlying backup or snapshot management system, available outside of the ISTM server implementation, at the operating system level. The backup support for the storage can be implemented in a hierarchical fashion, with uniquely identified containers of point-in-time copies (snapshot sets) of miscellaneous content (snapshots) within a set.

The data model includes the following persisted data:

SnapshotList: A collection of known snapshots for the existing LUN. Each snapshot is abstracted as an instance of the **ISnapshot** interface (section 3.36). Each element of this collection is a **Snapshot** data object.

Snapshot: A data model object is maintained for each snapshot to support the management of the snapshot. It includes the following elements:

Description: A user-friendly description of the snapshot.

ExportedWTD: The index within the collection managed by the disk manager of the LUN through which a point-in-time version of the content of the LUN is exposed.

Id: A GUID that uniquely identifies the snapshot.

OrigWTD: The index within the collection managed by the disk manager of the LUN of which this snapshot is a point-in-time version.

Persistent: A Boolean value that specifies whether the snapshot is persistent across system restarts.

Status: The status of the snapshot operation, which SHOULD be one or more of the following values:

Name/Value	Description
SnapshotStatusIdle 0	No snapshot operation is active.
SnapshotStatusExported 1	The snapshot is exposed as a VHD disk.
SnapshotStatusLocallyMounted 2	The snapshot is surfaced as a LUN on the iSCSI Software Target computer for read-only access.
SnapshotStatusRollbackInProgress 4	The snapshot is in the process of being restored to disk.

TimeStamp: The time stamp of the snapshot.

VSSSnapshotsetId: A value that uniquely identifies the relationship between the snapshots managed by the ISTM server and the snapshot sets managed by the underlying snapshot or backup storage mechanism.

VSSSnapshotId: A value that uniquely identifies the relationship between a snapshot managed by the ISTM server and the snapshot managed by the storage infrastructure within the set.

3.1.1.6 iSNS Management Data Model

The following diagram shows some of the relationships between actors in the ISTM server for the iSNS management data model.

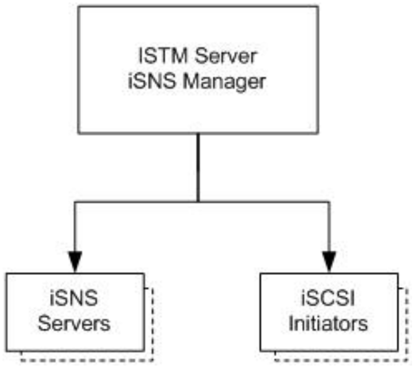


Figure 15: ISTM server iSNS manager

A data model is maintained by the iSNS management subsystem to support the management of iSNS servers. It includes the following persisted data:

SnsServerList: A collection of data objects that identify iSNS servers. Each element of this collection is an **SnsServer** data object.

SnsServer: An object that identifies an iSNS server. It includes the following information:

A string containing the DNS name or IP address of the iSNS server.

A Boolean flag indicating whether the iSNS server was discovered on the network using the Dynamic Host Configuration Protocol (DHCP).

CachedInitiatorList: A collection of the iSCSI initiators that have been discovered by the ISTM server. The discovery mechanism includes, but it is not limited to, a cache of initiators that previously attempted to log in. Each element of this collection is a **CachedInitiator** data object.

CachedInitiator: A data object that includes the following elements:

InitiatorIQN: The IQN of the initiator.

LastLoginTimeStamp: The time stamp of the last login attempt by the initiator.

When an ISTM client makes a request for a cached iSCSI initiator, that information is obtained by querying the iSNS servers; thus, a persisted collection of cached iSCSI initiators is not required.

3.1.1.7 Target Management Data Model

The following diagram shows some of the relationships between actors in the ISTM server for the target management data model.

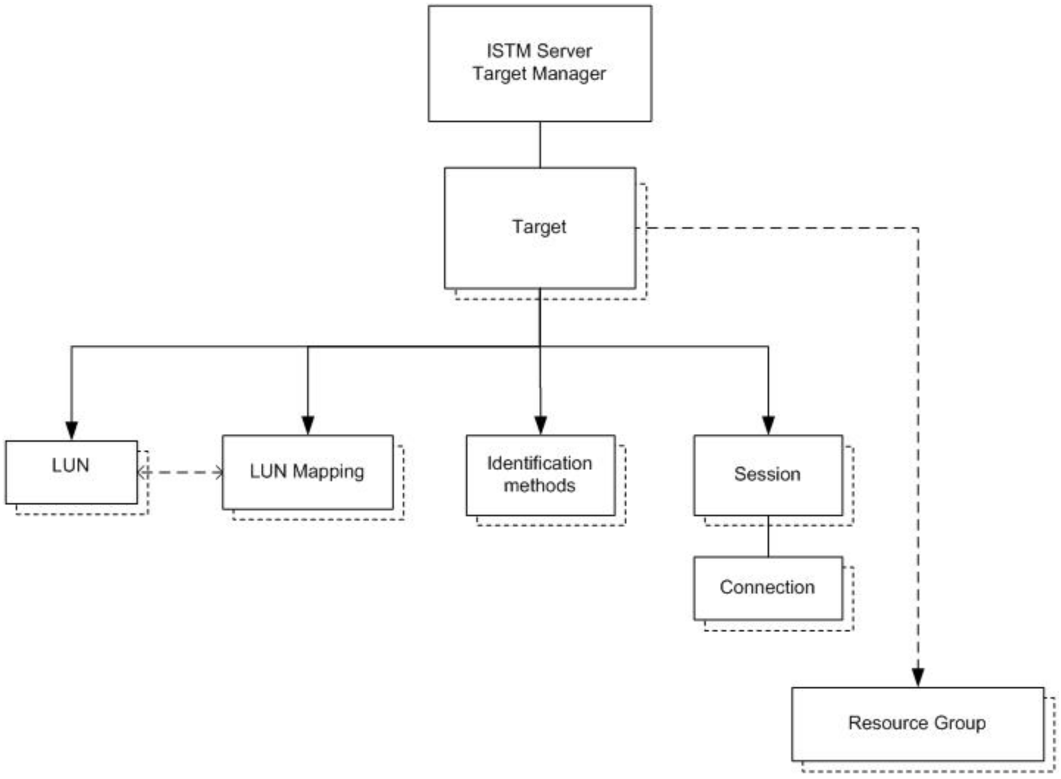


Figure 16: ISTM server target manager

The target manager maintains a collection of iSCSI targets, either directly or indirectly, by having access to the same collection that is maintained by the iSCSI Software Target implementation being managed.

The following persisted data is maintained by the ISTM server, or equivalently, by the underlying iSCSI target implementation.

HostList: A collection of **IHost** interface (section 3.21) objects for all iSCSI targets on the iSCSI Software Target. Each IHost object defines the following properties:

Note In an HA cluster configuration, offline target cluster resources SHOULD NOT be present in the **HostList** collection.

Guid: An identifier for the iSCSI target from the point of view of the ISTM server.

Enable: The state of the target, whether it is enabled or disabled.

Name: The name of the target, as specified by the ISTM client.

Description: A description for the target as specified by the ISTM client.

TargetIQN: The iSCSI qualified name (IQN) of the target.

Status: The status of the target; that is, whether the target is logged in or idle, as specified in the **HostStatus** enumeration (section 2.2.3.4).

LastLogIn: The last time that an iSCSI initiator logged in.

EnableCHAP: Whether Challenge-Handshake Authentication Protocol (CHAP) [RFC1994] [MS-CHAP] authentication is required for an iSCSI initiator to connect to the target.

CHAPUserName: The user name for CHAP authentication on the target.

CHAPSecret: The CHAP secret for authentication on the target.

EnableReverseCHAP: Whether CHAP **mutual authentication** is required for an iSCSI initiator to connect to the target.

ReverseCHAPUSername: The user name for CHAP mutual authentication on the target.

ReverseCHAPSecret: The CHAP secret for CHAP mutual authentication on the target.

TargetMaxRecvDataSegmentLength: The maximum receive-data segment length.

TargetFirstBurstLength: The first **burst length**.

TargetMaxBurstLength: The maximum burst length.

NumRecvBuffers: The number of receive buffers.

EnforceIdleTimeoutDetection: Whether idle timeout detection is enforced.

ListenAllNetworkAdapters: Whether the target listens on all network adapters.

The following data objects are maintained for targets in an HA cluster environment:

ResourceName: The resource name.

ResourceGroupName: The resource group name.

ResourceState: The cluster state of the target resource, which SHOULD be one of the values specified in [MS-CMRP] section 3.1.4.1.13.

IDMethodList: A collection of data objects that specify identification methods for the target. Each object contains the following elements:

IDMethodType: The target identification type, which specifies whether the host is identified by using a NetBIOS name, the DNS name, an IP address, a MAC address, or an IQN, as specified in the **IDMethod** enumeration (section 2.2.3.5).

IDMethodValue: The target identification address.

All identification methods for the target can be enumerated by an iterator exposed by the **IEnumIDMethod** interface (section 3.9).

In an HA cluster environment, the state of the target can be changed from online to offline and vice versa.

WTDiskLunMappingList: A collection of mappings of LUN numbers to virtual disks for the target. Each element contains a LUN identifier and a virtual disk index, as specified by the **LUN_MAPPING** structure (section 2.2.4.10). All LUN mappings for the target can be enumerated by an iterator exposed by the **IEnumWTDiskLunMapping** interface (section 3.20).

The relationship of an active session with the target is also maintained. The data model for the session is the same as that described in section 3.1.1.3, but it is limited to the session that is active on the target.

The relationship of iSCSI initiator portals with the target is also maintained. The data model for the portals is the same as that described in section 3.1.1.3, but it is limited to portals that are servicing storage requests for the target.

3.1.2 Timers

None.

3.1.3 Initialization

The ISTM server MUST perform the following initialization actions:

- Initialize and register for discoverability all COM interfaces as specified in [MS-DCOM].
- Instantiate all persistent data objects and object collections specified in the abstract data model (section 3.1.1).
- Initialize all persistent data objects and object collections from the state of the iSCSI target according to the system configuration.

The ISTM server MUST create the management interface objects shown in the following table, associate them with appropriate CLSID values (section 1.9) so that they are available to the ISTM client, and initialize their corresponding data model.

Subsystem	Interface	Data Model
Feature management	IWTGeneral	Section 3.1.1.1
Session management	ISessionManager	Section 3.1.1.2
Local device management	ILocalDeviceMgr	Section 3.1.1.3
LUN management	IWTDiskMgr	Section 3.1.1.4
Snapshot management	ISnapshotMgr	Section 3.1.1.5
iSNS management	ISnsMgr	Section 3.1.1.6
Target management	IHostMgr	Section 3.1.1.7

3.1.4 Message Processing Events and Sequencing Rules

This section specifies message processing events and sequencing rules that are not specific to a particular ISTM server interface.

3.1.4.1 Interface Associations

This section shows the associations between COM interfaces and each management subsystem of the iSCSI Software Target. Each interface has the **IUnknown** interface [MS-DCOM] at the beginning of its inheritance chain.

3.1.4.1.1 Feature Management Interfaces

The feature management subsystem uses the interfaces shown in the following table:

Interface	Section	Description
IStatusNotify	3.41	Used by the ISTM server to send status notifications to ISTM clients.
IWTGeneral	3.52	Used to maintain global settings and statuses.
IWTGeneral2	3.54	Extends the IWTGeneral interface to include information for HA cluster configurations.

3.1.4.1.2 Session Management Interfaces

The session management subsystem uses the interfaces shown in the following table:

Interface	Section	Description
IEnumConnection	3.5	Used to enumerate iSCSI connections.
IEnumSession	3.13	Used to enumerate iSCSI sessions.
IEnumSession2	3.14	Used to enumerate iSCSI sessions.
ISessionManager	3.34	Used to obtain information about iSCSI sessions.
ISessionManager2	3.35	Used to obtain information about iSCSI sessions and iSCSI connections.

3.1.4.1.3 Local Device Management Interfaces

The local device management subsystem uses the interfaces shown in the following table:

Interface	Section	Description
IDirectoryEnum	3.3	Used to enumerate the directories and files that exist in a volume.
IEnumDisk	3.6	Used to enumerate the disks in the iSCSI Software Target computer.<7>
IEnumDisk2	3.7	Extends the IEnumDisk interface with support for HA clusters.<8>
IEnumPortal	3.11	Used to enumerate the portals that exist on the iSCSI Software Target.
IEnumResourceGroup	3.12	Used to enumerate the resource groups that relate to the HA clustering configuration of the iSCSI Software Target computer.
IEnumVolume	3.17	Used to enumerate the volumes that have been configured on the iSCSI Software Target computer.
IEnumVolume2	3.18	Extends the IEnumVolume interface with support for sparse files and HA clusters.
ILocalDeviceMgr	3.30	Used to manage devices that are local to the iSCSI Software Target computer, including their configuration information.
ILocalDeviceMgr2	3.32	Extends the ILocalDeviceMgr interface with support for HA clusters.
ILocalDeviceMgr3	3.33	Extends the ILocalDeviceMgr2 interface with support for differencing VHD files.

3.1.4.1.4 LUN Management Interfaces

The LUN management subsystem uses the interfaces shown in the following table:

Interface	Section	Description
IEnumLMMountPoint	3.10	Used to enumerate the mount points associated with LUNs.
IEnumWTDisk	3.19	Used to enumerate the virtual disks that reside on the iSCSI Software Target or on a particular volume.
IWTDisk	3.43	Used to configure and maintain individual virtual disks.
IWTDisk2	3.44	Extends the IWTDisk interface with support for HA clusters.
IWTDisk3	3.45	Extends the IWTDisk2 interface with additional support for HA clusters and differencing VHD files.
IWTDiskMgr	3.46	Used to create and manage the LUNs and associated information that have been configured for the iSCSI Software Target.
IWTDiskMgr2	3.48	Extends the IWTDiskMgr interface with support for non-persistent virtual disk configurations.
IWTDiskMgr3	3.50	Extends the IWTDiskMgr2 interface with support for HA clusters.
IWTDiskMgr4	3.51	Extends the IWTDiskMgr3 interface with additional support for HA clusters.

3.1.4.1.5 Snapshot Management Interfaces

The snapshot management subsystem uses the interfaces shown in the following table:

Interface	Section	Description
IEnumSnapshot	3.15	Used to enumerate snapshots that have been created on the iSCSI Software Target.
ISnapshot	3.36	Used to manage a particular snapshot.
ISnapshotMgr	3.38	Used to manage disk backup and restore operations.

3.1.4.1.6 iSNS Management Interfaces

The iSNS management subsystem uses the interfaces shown in the following table:

Interface	Section	Description
IEnumCachedInitiator	3.4	Used to enumerate the iSCSI initiators that have been discovered by ISTM.
IEnumSnsServer	3.16	Used to enumerate the iSNS servers that are registered with the iSCSI Software Target.
ISnsMgr	3.40	Used to manage information about iSNS servers.

3.1.4.1.7 Target Management Interfaces

The target management subsystem uses the interfaces shown in the following table:

Interface	Section	Description
IEnumHost	3.8	Used to enumerate all the iSCSI targets that have been configured in the iSCSI Software Target.
IEnumIDMethod	3.9	Used to enumerate the identification methods for a particular iSCSI target.
IEnumPortal	3.11	Used to enumerate the portals that exist on the iSCSI Software Target.
IEnumSession2	3.14	Used to enumerate iSCSI sessions.
IEnumWTDiskLunMapping	3.20	Used to enumerate the LUNs that are assigned to a particular iSCSI target.
IHost	3.21	Used to manage a particular iSCSI target.
IHost2	3.23	Extends the IHost interface with support for HA clusters and non-persistent iSCSI targets.
IHost3	3.24	Extends the IHost2 interface with support for HA clusters.
IHostMgr	3.25	Used to manage information about iSCSI targets that initiators can connect to.
IHostMgr2	3.27	Extends the IHostMgr interface with support for non-persistent iSCSI targets.
IHostMgr3	3.29	Extends the IHostMgr2 interface with support for HA clusters.

3.1.4.2 Common Processing

This section specifies common processing details for ISTM server interface methods.

3.1.4.2.1 Check Preconditions

Many ISTM server methods check preconditions as their first action of processing. In general, the purpose of checking preconditions is to ensure that certain operational preconditions are met and that the method caller has appropriate access in order for the method to perform its processing.

The ISTM client can also check preconditions, in order to determine if the user application is able to perform its iSCSI Software Target management functions.

The specific preconditions can be system-dependent, and they include the following:

- Verify that the ISTM server target services that are needed to perform processing by the calling method are available and fully operational.<9>
- Verify the presence of HA clustering services are available. If they are present, their operational status is tested. Those are needed to perform specific processing in an HA cluster environment.<10>

- Verify that the calling ISTM client has authorization to perform the processing by the specific method.<11>

3.1.4.2.2 Enumerate Objects

Unless otherwise specified, the message processing defined in the following sections is common to the methods of all enumeration interfaces.

3.1.4.2.2.1 Next Enumeration Method

The **Next** enumeration method retrieves a specified number of items in a collection or the remaining items in a collection, whichever is smaller.<12> The *ulNumElements* parameter specifies the number of elements to retrieve, and the *pNumElementsReturned* parameter is used to return the actual number of elements that are retrieved. The *pElements* parameter, if specified, is used to return the items from the collection.

The Next method returns S_OK for success, S_FALSE if the requested number of items exceeds the limit of the collection, and failure if the enumeration is not initialized, or for invalid parameters.<13>

It SHOULD be possible to use the Next method to determine the number of enumerated objects that have not yet been retrieved. If the *ulNumElements* parameter is zero, the *pElements* parameter is NULL, and the *pNumElementsReturned* parameter is not NULL, the number of remaining items SHOULD be returned using the *pNumElementsReturned* parameter.

When there are no more objects to be retrieved, the server returns S_FALSE, with NULL in the *pElements* parameter and zero using the *pNumElementsReturned* parameter if it is not NULL.

3.1.4.2.2.2 Skip Enumeration Method

The **Skip** enumeration method instructs the ISTM server to skip over a certain number of enumerated objects. It does this by advancing the **Iterator** (section 3.1.1) the number of items specified in the *ulNumElements* parameter. If the skip action causes the enumerator to go past the end of the list, with the result that there are no more objects to be retrieved, the server returns S_FALSE.

3.1.4.2.2.3 Reset Enumeration Method

The **Reset** enumeration method sets the **Iterator** (section 3.1.1) of an enumeration back to the first item in the list.

The ISTM server SHOULD return an appropriate **HRESULT** (section 2.2) value [MS-ERREF] if the enumeration cannot be reset back to the first item in the list.<14>

Note This operation might not be meaningful for some implementations of an **Iterator**.

3.1.4.2.2.4 Clone Enumeration Method

The **Clone** enumeration method creates an interface object of the same type as the current enumeration interface. This new interface object provides access to the same data as the current interface object. The **Iterator** (section 3.1.1) of the new interface object is not related to the **Iterator** of the current interface object. The ISTM server returns the pointer to the new interface in the *ppNewEnum* parameter.

3.1.4.2.3 Exceptions

Unless otherwise specified, all methods MUST NOT throw exceptions.

3.1.4.2.4 Return Values

If an enumeration value is specified for a server method parameter, and the method is called with a value for that parameter defined as "Reserved" in section 2.2.3, the method SHOULD return an appropriate error, such as E_INVALIDARG.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 Common Client Details

This section provides details that are common to all ISTM client processing, regardless of interface.

3.2.1 Abstract Data Model

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

The following data object is common to all ISTM client interactions with ISTM server enumeration interfaces:

Iterator: The ISTM client uses a cursor mechanism to iterate through a collection of enumerated objects of a certain type on the ISTM server by invoking the **Next** method of an enumeration interface (section 3.1.4.2.2.1).

A data model is maintained by the ISTM client to support status notifications. It is described in the **IStatusNotify** server abstract data model (section 3.41.1).

3.2.2 Timers

None.

3.2.3 Initialization

An ISTM client SHOULD initialize by binding to ISTM server interfaces, as specified in [MS-DCOM] section 3.1.4. All other server interfaces can be discovered from the following interfaces:

- IHostMgr
- ILocalDeviceMgr
- ISessionManager
- ISnapshotMgr

- ISNSMgr
- IWTDiskMgr
- IWTGeneral

The **CLSID** values for these interfaces are defined in section 1.9.

3.2.4 Message Processing Events and Sequencing Rules

This section specifies message processing events and sequencing rules that apply generally to ISTM client interaction with the ISTM server, and are not specific to a particular interface.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

3.3 IDirectoryEnum Server Details

The **IDirectoryEnum** interface is used to enumerate the directories and files that exist in a volume.

3.3.1 Abstract Data Model

The **IDirectoryEnum** interface provides an abstraction of an iterator over a single level of a hierarchical file system. The file system path to iterate is specified to the server in the initial call to the **FindFirst** method (section 3.3.4.2). This initial call, if successful, SHOULD return the requested information for the first enumerated file system item. Subsequent calls to the **FindNext** method (section 3.3.4.3) SHOULD return the remaining directories or files in the level of the file system that was specified in the call to **FindFirst**, until the end of the enumeration is reached, or the enumeration is closed.

For more details concerning the abstract data organization for this interface, see the local device management data model for file data (section 3.1.1.3.4).

3.3.2 Timers

None.

3.3.3 Initialization

Local device manager initialization is specified in section 3.1.3.

3.3.4 Message Processing Events and Sequencing Rules

The **IDirectoryEnum** interface includes the following methods:

Method	Description
GetDriveString	Retrieves the set of disk drives in the system. Opnum: 3

Method	Description
FindFirst	Retrieves information about the first file system object in a given directory. Opnum: 4
FindNext	Retrieves information about subsequent file system objects in the directory. Opnum: 5
FindClose	Ends the enumeration of the directory. Opnum: 6

3.3.4.1 GetDriveString (Opnum 3)

The **GetDriveString** method MAY retrieve a list of drives that are available on the iSCSI Software Target computer.<15>

```
HRESULT GetDriveString(
        [out] BSTR * pbstrDriveString);
```

pbstrDriveString: A pointer to a **BSTR** (section 2.2) that returns the list of drives.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.3.4.2 FindFirst (Opnum 4)

The **FindFirst** method retrieves information about the first file system object in a given directory.

```
HRESULT FindFirst(
        [in] BSTR bstrSearchPath,
        [in, out] DIRECTORY_DATA * pDirectoryData);
```

bstrSearchPath: The directory of interest.

pDirectoryData: A pointer to a **DIRECTORY_DATA** structure (section 2.2.4.3) that returns information about a file system object.

Return Values:

The following value is possible:

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server finds the first file system object in the directory specified by the *bstrSearchPath*, parameter, gathers information about the object according to the abstract data model in section 3.1.1.3.4, and returns that information to the client using the *pDirectoryData* parameter.

The ISTM server SHOULD perform additional validation on the state of the file system enumeration, such as validating that there was no enumeration previously started and not closed. The ISTM server might be tolerant to such a usage pattern in an implementation-specific manner.<16>

3.3.4.3 FindNext (Opnum 5)

The **FindNext** method retrieves information about file system objects using, as a starting point, the enumeration that was started by a call to the **FindFirst** method (section 3.3.4.2), and which is continued with subsequent calls to this method.

```
HRESULT FindNext(
        [in, out] DIRECTORY_DATA * pDirectoryData);
```

pDirectoryData: A pointer to a **DIRECTORY_DATA** structure (section 2.2.4.3) that returns information about a file system object.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NO_MORE_FILES 0x80070012	There are no more items to enumerate.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The ISTM server enumerates the next file system object in the directory that was specified by the *bstrSearchPath* parameter during the call to the **FindFirst** method. The file system object that is found depends on the objects that were found during the call to the **FindFirst** method and subsequent calls to this method. The order in which file system items (child directories or files) are returned is implementation-specific.

The server gathers information about the object according to the abstract data model in section 3.1.1.3.4, and returns that information to the client by using the *pDirectoryData* parameter.

If all file system objects in the directory have already been retrieved by previous calls to this method, or by the call to the **FindFirst** method, the server SHOULD return ERROR_NO_MORE_FILES.

Note If the **FindFirst** method was not called before this method is called, the server SHOULD return an error.

3.3.4.4 FindClose (Opnum 6)

The **FindClose** method ends the enumeration of the directory that was started by a call to the **FindFirst** method (section 3.3.4.2).

```
HRESULT FindClose();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

The ISTM server releases any resources associated to the enumeration, and, prepares the directory enumeration for subsequent use.

3.3.5 Timer Events

None.

3.3.6 Other Local Events

None.

3.4 IEnumCachedInitiator Server Details

The **IEnumCachedInitiator** interface is used to enumerate the iSCSI initiators that have been discovered by ISTM.

3.4.1 Abstract Data Model

The **IEnumCachedInitiator** interface provides an abstraction of an iterator over iSCSI initiators that exist in the iSNS manager. The collection of initiators being returned consists of initiators that have logged onto iSCSI servers in the past and initiators that were discovered using an iSNS server that has been configured for ISTM.

For more details concerning the abstract data organization for this interface, see the iSNS management data model (section 3.1.1.6).

3.4.2 Timers

The collection of iSCSI initiators that have logged on in the past has each entry time stamped. An entry is removed from the collection 7 days after it has been added, unless the iSCSI initiator logs back on to the iSCSI server. The portion of the collection that encompasses iSCSI initiators that were discovered through an iSNS server does not expire.

3.4.3 Initialization

The **IEnumCachedInitiator** interface is initialized by the iSNS management method **EnumCachedInitiators** (section 3.40.4.4). iSNS manager initialization is specified in section 3.1.3.

3.4.4 Message Processing Events and Sequencing Rules

The **IEnumCachedInitiator** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.4.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about a cached initiator.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] CACHED_INITIATOR * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. The value SHOULD NOT be greater than one.

pElements: A pointer to a **CACHED_INITIATOR** structure (section 2.2.4.1) that returns the initiator information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.

Return value/code	Description
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1

3.4.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the cached initiator enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.4.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the cached initiator enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.4.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumCachedInitiator ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumCachedInitiator** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.4.5 Timer Events

None.

3.4.6 Other Local Events

None.

3.5 IEnumConnection Server Details

The **IEnumConnection** interface is used to enumerate iSCSI connections.

3.5.1 Abstract Data Model

The **IEnumConnection** interface provides an abstraction of an iterator over iSCSI connections. The scope of the enumeration is given by the session identifier used to obtain the **IEnumConnection** interface.

For more details concerning the abstract data organization for this interface, see the session management data model for iSCSI connections (section 3.1.1.2.2).

3.5.2 Timers

None.

3.5.3 Initialization

The **iEnumConnection** interface is initialized by the session management method **EnumConnections** (section 3.35.4.2). Session manager initialization is specified in section 3.1.3.

3.5.4 Message Processing Events and Sequencing Rules

The **IEnumConnection** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.5.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about an iSCSI connection.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] CONNECTION * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to a **CONNECTION** structure (section 2.2.4.2) that returns the connection information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized, or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.5.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the iSCSI connection enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate, or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.5.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the iSCSI connection enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.5.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumConnection ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumConnection** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.5.5 Timer Events

None.

3.5.6 Other Local Events

None.

3.6 IEnumDisk Server Details

The **IEnumDisk** interface MAY be used to enumerate the disks in the iSCSI Software Target computer.<17>

3.6.1 Abstract Data Model

The **IEnumDisk** interface provides an abstraction of an iterator of the operating system abstraction of the disk drives connected to a computer.

3.6.2 Timers

None.

3.6.3 Initialization

Local device manager initialization is specified in section 3.1.3.

3.6.4 Message Processing Events and Sequencing Rules

The **IEnumDisk** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.6.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about a disk.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] DISK * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. The value SHOULD NOT be greater than one.

pElements: A pointer to a **DISK** structure (section 2.2.4.4) that returns the disk information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE	There are no more items to enumerate.

Return value/code	Description
0x00000001	
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized, or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.6.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the disk enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate, or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.6.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the disk enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.6.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumDisk ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumDisk** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.6.5 Timer Events

None.

3.6.6 Other Local Events

None.

3.7 IEnumDisk2 Server Details

The **IEnumDisk2** interface extends the **IEnumDisk** interface (section 3.6) with support for HA clusters.<18>

3.7.1 Abstract Data Model

The **IEnumDisk2** interface provides an abstraction of an iterator of the operating system abstraction of the disk drives connected to a computer that is a node in an HA cluster.

3.7.2 Timers

None.

3.7.3 Initialization

Local device manager initialization is specified in section 3.1.3.

3.7.4 Message Processing Events and Sequencing Rules

The **IEnumDisk2** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.7.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about a disk.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] DISK2 * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. The value SHOULD NOT be greater than one.

pElements: A pointer to the **DISK2** structure (section 2.2.4.5) that returns the disk information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE	There are no more items to enumerate.

Return value/code	Description
0x00000001	
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized, or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.7.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the disk enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate, or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.7.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the disk enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.7.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumDisk2 ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumDisk2** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.7.5 Timer Events

None.

3.7.6 Other Local Events

None.

3.8 IEnumHost Server Details

The **IEnumHost** interface is used to enumerate all the iSCSI targets that have been configured in the iSCSI Software Target. The enumerated items are represented by instances of the **IHost** interface (section 3.21).

3.8.1 Abstract Data Model

The **IEnumHost** interface provides an abstraction of an iterator of the collection of iSCSI targets . In an HA cluster, the scope of the collection depends on the method being used to retrieve the **IEnumHost** interface.

For more details concerning the abstract data organization for this interface, see the target management data model (section 3.1.1.7).

3.8.2 Timers

None.

3.8.3 Initialization

Target manager initialization is specified in section 3.1.3.

3.8.4 Message Processing Events and Sequencing Rules

The **IEnumHost** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.8.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that specifies an iSCSI target.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] IHost ** pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to an **IHost** interface (section 3.21) that returns the target information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

This method SHOULD return an interface pointer of type IHost, even if the implementation supports other interfaces, such as **IHost2** (section 3.23) or **IHost3** (section 3.24).<19>

The interface pointers returned by the Next method are proxies to the iSCSI target elements of the collection maintained by the target manager. Those proxies are identical to targets retrieved through other methods, such as **GetHost** (section 3.25.4.3) and **GetHostByTargetIQN** (section 3.25.4.4).

3.8.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the target enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.8.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the target enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.8.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumHost ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumHost** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.8.5 Timer Events

None.

3.8.6 Other Local Events

None.

3.9 IEnumIDMethod Server Details

The **IEnumIDmethod** is used to enumerate the identification methods that are configured to allow access to a particular iSCSI target. The enumerated items are represented by instances of the **ID** structure (section 2.2.4.7).

3.9.1 Abstract Data Model

The **IEnumIDMethod** interface provides an abstraction of an iterator over the identification methods for the iSCSI initiators that are configured for the target.

For more details concerning the abstract data organization for this interface, see the target management data model (section 3.1.1.7).

3.9.2 Timers

None.

3.9.3 Initialization

Target manager initialization is specified in section 3.1.3.

3.9.4 Message Processing Events and Sequencing Rules

The **IEnumIDmethod** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.9.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that specifies an identification method.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] ID * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to an **ID** structure (section 2.2.4.7).

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.9.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the identification method enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.9.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the identification method enumeration back to the first item

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.9.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumIDMethod ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumIDMethod** interface pointer that returns the created object. This parameter MUST NOT be NULL

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.9.5 Timer Events

None.

3.9.6 Other Local Events

None.

3.10 IEnumLMMountPoint Server Details

The **IEnumLMMountPoint** interface is used to enumerate the mount points associated with a LUN when the LUN has been locally mounted (see section 3.43.4.35).

3.10.1 Abstract Data Model

The **IEnumLMMountPoint** interface provides an abstraction of an iterator over paths that identify mount points. The scope of the enumeration is given by the volumes that reside completely or partially in a disk whose plug and play identifier contains the currently mounted LUN.

For more details concerning the abstract data organization for this interface, see the LUN management data model (section 3.1.1.4).

3.10.2 Timers

None.

3.10.3 Initialization

LUN manager initialization is specified in section 3.1.3.

3.10.4 Message Processing Events and Sequencing Rules

The **IEnumLMMountPoint** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items . Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.10.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about a mount point.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] MOUNT_POINT * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. The value SHOULD NOT be greater than one.

pElements: A pointer to **MOUNT_POINT** structure (section 2.2.4.121) that returns the mount point information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.10.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the mount point enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. The value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.10.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the mount point enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.10.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumLMMountPoint ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumLMMountPoint** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.10.5 Timer Events

None.

3.10.6 Other Local Events

None.

3.11 IEnumPortal Server Details

The **IEnumPortal** interface is used to enumerate the portals that exist on the iSCSI Software Target.

3.11.1 Abstract Data Model

The **IEnumPortal** interface provides an abstraction of an iterator over the network portals that are available on the iSCSI Software Target. The scope of the collection of portals being returned is determined by the method used to obtain the **IEnumPortal** interface.

For more details concerning the abstract data organization for this interface, see the local device management data model for portal data (section 3.1.1.3.5).

3.11.2 Timers

None.

3.11.3 Initialization

Local device manager initialization is specified in section 3.1.3.

3.11.4 Message Processing Events and Sequencing Rules

The **IEnumPortal** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.11.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about a portal, as defined in [RFC3720].

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] PORTAL * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to a **PORTAL** structure (section 2.2.4.13) that returns the portal information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized, or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.11.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the portal enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate, or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.11.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the portal enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.11.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumPortal ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumPortal** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.11.5 Timer Events

None.

3.11.6 Other Local Events

None.

3.12 IEnumResourceGroup Server Details

The **IEnumResourceGroup** interface is used to enumerate the resource groups that relate to the HA clustering configuration of the iSCSI Software Target computer.

3.12.1 Abstract Data Model

The **IEnumResourceGroup** interface provides an abstraction of an iterator over the resource groups that are available in the HA clustering configuration. The scope of the collection of resource groups being returned is determined by the method used to obtain the **IEnumResourceGroup** interface (see sections 3.32.4.4 and 3.32.4.5).

For more details concerning the abstract data organization for this interface, see the local device management data model for resource group data (section 3.1.1.3.7).

3.12.2 Timers

None.

3.12.3 Initialization

Local device manager initialization is specified in section 3.1.3.

3.12.4 Message Processing Events and Sequencing Rules

The **IEnumResourceGroup** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.12.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about a resource group.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] RESOURCE_GROUP * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to a **RESOURCE_GROUP** structure (section 2.2.4.143) that returns the resource group information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.12.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the resource group enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. The value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.12.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the resource group enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.12.4.4 Clone (Opnum 6)

The **Clon**e method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumResourceGroup ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumResourceGroup** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.12.5 Timer Events

None.

3.12.6 Other Local Events

None.

3.13 IEnumSession Server Details

The **IEnumSession** interface is used to enumerate iSCSI sessions, providing minimal information about the IQN identifiers of initiators and targets.

3.13.1 Abstract Data Model

The **IEnumSession** interface provides an abstraction of an iterator over the iSCSI sessions that are currently active on the iSCSI Software Target. The scope of the collection of sessions being returned is determined by the method used to obtain the **IEnumSession** interface.

For more details concerning the abstract data organization for this interface, see the session management data model for iSCSI session data (section 3.1.1.2.1).

3.13.2 Timers

None.

3.13.3 Initialization

Session manager initialization is specified in section 3.1.3.

3.13.4 Message Processing Events and Sequencing Rules

The **IEnumSession** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.13.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about an iSCSI session.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] SESSION * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to a **SESSION** structure (section 2.2.4.15) that returns the session information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized, or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.13.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the iSCSI session enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate, or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.13.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the iSCSI session enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.13.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumSession ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumSession** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.13.5 Timer Events

None.

3.13.6 Other Local Events

None.

3.14 IEnumSession2 Server Details

The **IEnumSession2** interface is used to enumerate iSCSI sessions, providing extended iSCSI protocol-specific information.

3.14.1 Abstract Data Model

The **IEnumSession2** interface provides an abstraction of an iterator over the iSCSI sessions that are currently active on the iSCSI Software Target. The scope of the collection of sessions being returned is determined by the method used to obtain the **IEnumSession2** interface.

For more details concerning the abstract data organization for this interface, see the session management data model for iSCSI session data (section 3.1.1.2.1).

3.14.2 Timers

None.

3.14.3 Initialization

Session manager initialization is specified in section 3.1.3.

3.14.4 Message Processing Events and Sequencing Rules

The **IEnumSession2** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.14.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about an iSCSI session.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] SESSION2 * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to a **SESSION2** structure (section 2.2.4.16) that returns the session information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized, or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.14.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the iSCSI session enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate, or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.14.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the iSCSI session enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.14.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumSession2 ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumSession2** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.14.5 Timer Events

None.

3.14.6 Other Local Events

None.

3.15 IEnumSnapshot Server Details

The **IEnumSnapshot** interface is used to enumerate snapshots of LUNs that have been created on the iSCSI Software Target.

3.15.1 Abstract Data Model

The **IEnumSnapshot** interface provides an abstraction of an iterator over the snapshots that exist in the snapshot manager. The scope of the collection of snapshots being returned is the entire set.

For more details concerning the abstract data organization for this interface, see the snapshot management data model (section 3.1.1.5).

3.15.2 Timers

None.

3.15.3 Initialization

The **IEnumSnapshot** interface is initialized by the snapshot management **EnumSnapshots** method (section 3.38.4.1). Snapshot manager initialization is specified in section 3.1.3.

3.15.4 Message Processing Events and Sequencing Rules

The **IEnumSnapshot** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.15.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that can be used to manage a snapshot.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] ISnapshot ** pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to the **ISnapshot** interface pointer (section 3.36) that is returned.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized, or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

The interface pointers returned by the Next method are proxies to the snapshot elements of the collection maintained by the snapshot manager. Those proxies are identical to snapshots retrieved through other methods, such as **GetSnapshot** (section 3.38.4.2).

3.15.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the snapshot enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate, or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.15.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the snapshot enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.15.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumSnapshot ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumSnapshot** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.15.5 Timer Events

None.

3.15.6 Other Local Events

None.

3.16 IEnumSnsServer Server Details

The **IEnumSnsServer** interface is used to enumerate the iSNS servers that are registered with the iSCSI Software Target.

3.16.1 Abstract Data Model

The **IEnumSnsServer** interface provides an abstraction of an iterator over iSNS servers. The scope of the collection being returned is the set of iSNS servers that have been registered with the iSCSI Software Target, including those that were discovered by the Dynamic Host Configuration Protocol (DHCP).

For more details concerning the abstract data organization for this interface, see the iSNS management data model (section 3.1.1.6).

3.16.2 Timers

None.

3.16.3 Initialization

The **IEnumSnsServer** interface is initialized by the iSNS management method **EnumSnsServers** (section 3.40.4.1). iSNS manager initialization is specified in section 3.1.3.

3.16.4 Message Processing Events and Sequencing Rules

The **IEnumSnsServer** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.16.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about an iSNS server.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] SNS_SERVER * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to an **SNS_SERVER** structure (section 2.2.4.18) that returns the iSNS server information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.16.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the iSNS server enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.16.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the iSNS server enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.16.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumSnsServer ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumSnsServer** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.16.5 Timer Events

None

3.16.6 Other Local Events

None.

3.17 IEnumVolume Server Details

The **IEnumVolume** interface is used to enumerate the volumes that are present on the iSCSI Software Target computer and have been configured for use by the iSCSI Software Target.

3.17.1 Abstract Data Model

The **IEnumVolume** interface provides an abstraction of an iterator over the volumes available to the iSCSI Software Target for the creation of LUNs. The scope of the collection of volumes being returned is global for the computer.

For more details concerning the abstract data organization for this interface, see the local device management data model for volume data (section 3.1.1.3.2).

3.17.2 Timers

None.

3.17.3 Initialization

Local device manager initialization is specified in section 3.1.3.

3.17.4 Message Processing Events and Sequencing Rules

The **IEnumVolume** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.17.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about a volume.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] VOLUME * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to a **VOLUME** structure (section 2.2.4.19) that returns the volume information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized, or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.17.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the volume enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate, or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.17.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the volume enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.17.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumVolume ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumVolume** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.17.5 Timer Events

None.

3.17.6 Other Local Events

None.

3.18 IEnumVolume2 Server Details

The **IEnumVolume2** interface extends the **IEnumVolume** interface (section 3.17) with support for sparse files and HA clusters.

3.18.1 Abstract Data Model

The **IEnumVolume2** interface provides an abstraction of an iterator over the volumes available to the iSCSI Software Target for the creation of LUNs when running in an HA cluster configuration. The scope of the collection of volumes being returned is limited to those based on clustered disks.

For more details concerning the abstract data organization for this interface, see the local device management data model for volume data (section 3.1.1.3.2).

3.18.2 Timers

None.

3.18.3 Initialization

Local device manager initialization is specified in section 3.1.3.

3.18.4 Message Processing Events and Sequencing Rules

The **IEnumVolume2** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.18.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that contains information about a volume.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] VOLUME2 * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to a **VOLUME2** structure (section 2.2.4.20) that returns the volume information.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized, or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.18.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the volume enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate, or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.18.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the volume enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.18.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumVolume2 ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumVolume2** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.18.5 Timer Events

None.

3.18.6 Other Local Events

None.

3.19 IEnumWTDisk Server Details

The **IEnumWTDisk** interface is used to enumerate all the virtual disks that exist on the iSCSI Software Target or those that are associated with a particular volume on the iSCSI Software Target. The enumerated virtual disks are represented by **IWTDisk** interface objects (sections 3.43, 3.44, and 3.45), which are referred to in this section as **WTDisk** items.

3.19.1 Abstract Data Model

The **IEnumWTDisk** interface provides an abstraction of an iterator over the virtual disks that are available in the scope specified by the method used to obtain the **IEnumWTDisk** interface (sections 3.46.4.1, 3.46.4.6, 3.46.4.7, and 3.51.4.1).

For more details concerning the abstract data organization for this interface, see the LUN management data model (section 3.1.1.4).

3.19.2 Timers

None.

3.19.3 Initialization

LUN manager initialization is specified in section 3.1.3.

3.19.4 Message Processing Events and Sequencing Rules

The **IEnumWTDisk** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated **WTDisk** item. Opnum: 3
Skip	Skips a specified number of enumerated **WTDisk** items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.19.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated **WTDisk** item.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] IWTDisk ** pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of **WTDisk** items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to an **IWTDisk** interface pointer that provides access to the **WTDisk** item.

pNumElementsReturned: A pointer to a value that returns the number of **WTDisk** items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

The interface pointer that is returned has a data type of **IWTDisk**, even if the implementation supports other interfaces, such **IWTDisk2** (section 3.44) and **IWTDisk3** (section 3.45).

The interface pointers returned by the **Next** method are proxies to the LUN elements of the collection maintained by the LUN manager. Those proxies are identical to LUNs retrieved through other methods, such as **GetWTDisk** (section 3.46.4.5).

3.19.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of **WTDisk** items in the enumerator.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE	There are no more items to enumerate or the number of items to skip went past the

Return value/code	Description
0x00000001	boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.19.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the **WTDisk** items enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.19.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumWTDisk ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumWTDisk** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.19.5 Timer Events

None.

3.19.6 Other Local Events

None.

3.20 IEnumWTDiskLunMapping Server Details

The **IEnumWtDIskLunMapping** interface enumerates mappings of LUN numbers [SAM-3] with virtual disks that are associated with a specified target. The enumerated items are represented by instances of the **LUN_MAPPING** structure (section 2.2.4.10).

3.20.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the LUN management data model (section 3.1.1.4).

3.20.2 Timers

None.

3.20.3 Initialization

LUN manager initialization is specified in section 3.1.3.

3.20.4 Message Processing Events and Sequencing Rules

The **IEnumWtDIskLunMapping** interface includes the following methods:

Method	Description
Next	Retrieves an enumerated item. Opnum: 3
Skip	Skips a specified number of enumerated items. Opnum: 4
Reset	Sets the enumeration back to the first item. Opnum: 5
Clone	Creates an interface object for the same enumeration. Opnum: 6

3.20.4.1 Next (Opnum 3)

The **Next** method retrieves an enumerated item that specifies a LUN mapping.

```
HRESULT Next(
        [in] ULONG ulNumElements,
        [in, out] LUN_MAPPING * pElements,
        [out] ULONG * pNumElementsReturned);
```

ulNumElements: The number of items to be retrieved. This value SHOULD NOT be greater than one.

pElements: A pointer to a **LUN_MAPPING** structure (section 2.2.4.10) that returns the LUN mapping.

pNumElementsReturned: A pointer to a value that returns the number of items retrieved using the *pElements* parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	There are no more items to enumerate.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The enumerator is not initialized or the caller requested more than one item.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Next** method is specified in section 3.1.4.2.2.1.

3.20.4.2 Skip (Opnum 4)

The **Skip** method skips a specified number of items in the LUN mapping enumeration.

```
HRESULT Skip(
        [in] ULONG ulNumElements);
```

ulNumElements: The number of items to skip. This value MUST be greater than zero.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

Return value/code	Description
S_FALSE 0x00000001	There are no more items to enumerate or the number of items to skip went past the boundary of the enumerator.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Skip** method is specified in section 3.1.4.2.2.2.

3.20.4.3 Reset (Opnum 5)

The **Reset** method sets the **Iterator** (section 3.1.1) of the LUN mapping enumeration back to the first item.

```
HRESULT Reset();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Reset** method is specified in section 3.1.4.2.2.3.

3.20.4.4 Clone (Opnum 6)

The **Clone** method creates an interface object for the same enumeration as the current interface object.

```
HRESULT Clone(
        [in] IEnumWTDiskLunMapping ** ppNewEnum);
```

ppNewEnum: A pointer to an **IEnumWTDiskMapping** interface pointer that returns the created object. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Processing of the **Clone** method is specified in section 3.1.4.2.2.4.

3.20.5 Timer Events

None.

3.20.6 Other Local Events

None.

3.21 IHost Server Details

The **IHost** interface is used to manage a particular iSCSI target.

3.21.1 Abstract Data Model

The **IHost** interface provides an abstraction of an iSCSI target. It is created by the target manager on demand from an ISTM client. The client can add a LUN to a specific target or remove a particular LUN or all LUN objects from a particular target.

For more details concerning the abstract data organization for this interface, see the target management data model (section 3.1.1.7).

3.21.2 Timers

None.

3.21.3 Initialization

Target manager initialization is specified in section 3.1.3.

3.21.4 Message Processing Events and Sequencing Rules

The **IHost** interface includes the following methods:

Method	Description
[propget]Guid	Returns the identifier of the target. Opnum: 3
[propget]Enable	Returns the enabled state of the target. Opnum: 4
[propput]Enable	Sets the enabled state of the target. Opnum: 5
[propget]Name	Returns the iSCSI name of the target. Opnum: 6
[propput]Name	Sets the iSCSI name of the target. Opnum: 7
[propget]Description	Returns the description of the target. Opnum: 8

Method	Description
[propput]Description	Sets the description of the target. Opnum: 9
[propget]TargetIQN	Returns the IQN of the target. Opnum: 10
[propput]TargetIQN	Sets the IQN of the target. Opnum: 11
[propget]Status	Returns the connected status of the target. Opnum: 12
[propput]Status	Sets the connected status of the target. Opnum: 13
[propget]LastLogIn	Returns the time stamp of the last login on the target. Opnum: 14
[propput]LastLogIn	Sets the time stamp of the last login on the target. Opnum: 15
[propget]EnableCHAP	Returns whether CHAP authentication is required for an initiator to connect to the target. Opnum: 16
[propput]EnableCHAP	Sets whether CHAP authentication is required for an initiator to connect to the target. Opnum: 17
[propget]CHAPUserName	Returns the user name for CHAP authentication on the target. Opnum: 18
[propput]CHAPUserName	Sets the user name for CHAP authentication on the target. Opnum: 19
[propget]CHAPSecret	Returns the CHAP secret for CHAP authentication on the target. Opnum: 20
[propput]CHAPSecret	Sets the CHAP secret for CHAP authentication on the target. Opnum: 21
[propget]EnableReverseCHAP	Returns whether CHAP mutual authentication is required for an initiator to connect to the target. Opnum: 22
[propput]EnableReverseCHAP	Sets whether CHAP mutual authentication is required for an initiator to connect to the target. Opnum: 23
[propget]ReverseCHAPUserName	Returns the user name for CHAP mutual authentication on

Method	Description
	the target. Opnum: 24
[propput]ReverseCHAPUserName	Sets the user name for CHAP mutual authentication on the target. Opnum: 25
[propget]ReverseCHAPSecret	Returns the CHAP secret for CHAP mutual authentication on the target. Opnum: 26
[propput]ReverseCHAPSecret	Sets the CHAP secret for CHAP mutual authentication on the target. Opnum: 27
[propget]TargetMaxRecvDataSegmentLength	Returns the maximum receive-data segment length for the target. Opnum: 28
[propput]TargetMaxRecvDataSegmentLength	Sets the maximum receive-data segment length for the target. Opnum: 29
[propget]TargetFirstBurstLength	Returns the first burst length for the target. Opnum: 30
[propput]TargetFirstBurstLength	Sets the first burst length for the target. Opnum: 31
[propget]TargetMaxBurstLength	Returns the maximum burst length for the target. Opnum: 32
[propput]TargetMaxBurstLength	Sets the maximum burst length for the target. Opnum: 33
[propget]NumRecvBuffers	Returns the number of receive buffers for the target. Opnum: 34
[propput]NumRecvBuffers	Sets the number of receive buffers for the target. Opnum: 35
[propget]EnforceIdleTimeoutDetection	Returns whether idle timeout detection is enforced on the target. Opnum: 36
[propput]EnforceIdleTimeoutDetection	Sets whether idle timeout detection is enforced on the target. Opnum: 37
[propget]ListenAllNetworkAdapters	Returns whether the target listens on all network adapters. Opnum: 38
[propput]ListenAllNetworkAdapters	Sets whether the target listens on all network adapters.

Method	Description
	Opnum: 39
[propget]Processor	Returns the physical processor that is designated as the processing resource for requests directed to the target. Opnum: 40
[propput]Processor	Sets the processor for the target. Opnum: 41
EnumIDs	Returns an enumeration of identification methods for the iSCSI initiators that are configured for the target. Opnum: 42
AddID	Adds a specified identification method for an iSCSI initiator to the target. Opnum: 43
RemoveID	Removes a specified identification method for an iSCSI initiator from the target. Opnum: 44
RemoveAllIDs	Removes all identification methods from the target. Opnum: 45
EnumWTDiskLunMappings	Returns an enumeration of the mappings of LUN numbers to virtual disks that are assigned to the target. Opnum: 46
AddWTDisk	Adds a specified virtual disk assignment to the target. Opnum: 47
RemoveWTDisk	Removes a specified virtual disk assignment from the target. Opnum: 48
RemoveAllWTDisks	Removes all virtual disk assignments from the target. Opnum: 49
SetWTDiskLunMapping	Sets the LUN mapping for the specified virtual disk. Opnum: 50
EnumPortals	Returns an enumeration of the portals that are configured to service iSCSI initiator requests for the target. Opnum: 51
AddPortal	Adds a specified portal to the target. Opnum: 52
RemovePortal	Removes a specified portal from the target. Opnum: 53
RemoveAllPortals	Removes all portals from the target. Opnum: 54

Method	Description
SetCHAPInfoByBlob	Sets CHAP information for the target by using the data in a specified encrypted **BLOB**. Opnum: 55
Save	Saves target properties to persistent storage. Opnum: 56
ValidateChapSecret	Validates the specified CHAP secret. Opnum: 57
ValidateReverseChapSecret	Validates the specified mutual CHAP secret. Opnum: 58
CheckTargetRedirect	Checks whether redirection of the target is required. Opnum: 59

3.21.4.1 [propget]Guid (Opnum 3)

The **[propget]Guid** method returns the identifier of the target.

```
[propget] HRESULT Guid(
          [out] [retval] GUID * pGuid);
```

pGuid: A pointer to a **GUID** structure (section 2.2) that returns the identifier for the target. This value MUST be unique to the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Guid** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server

SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **Guid** data element (section 3.1.1.7) of the target by using the *pGuid* parameter.

3.21.4.2 [propget]Enable (Opnum 4)

The **[propget]Enable** method returns the enabled state of the target.

```
[propget] HRESULT Enable(
        [out] [retval] boolean * pEnable);
```

pEnable: A pointer to a Boolean value that returns the enabled state of the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Enable** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **Enable** data element (section 3.1.1.7) of the target by using the *pEnable* parameter.

3.21.4.3 [propput]Enable (Opnum 5)

The **[propput]Enable** method Sets the enabled state of the target.

```
[propput] HRESULT Enable(
        [in] boolean bEnable);
```

bEnable: A Boolean value that specifies the enabled state of the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]Enable** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Sets the value of the **Enable** data element (section 3.1.1.7) of the target from the *bEnable* parameter.
- As a side effect of the transition of the **Enable** data element from true to false, the server SHOULD disconnect any active session against the target.

3.21.4.4 [propget]Name (Opnum 6)

The **[propget]Name** method returns the iSCSI name of the target.

```
[propget] HRESULT Name(
        [out] [retval] BSTR * pName);
```

pName: A pointer to a string that returns the iSCSI name of the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Name** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

On receiving this call, the server returns the value of the **Name** data element (section 3.1.1.7) of the target by using the *pName* parameter.

3.21.4.5 [propput]Name (Opnum 7)

The **[propput]Name** method sets the iSCSI name of the target.

```
[propput] HRESULT Name(
        [in] BSTR Name);
```

Name: The iSCSI name of the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_DATA 0x8007000d	The target name is of an invalid format.
ERROR_FILE_EXISTS 0x80070050	The target name is the same of an already existing target.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]Name** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Checks whether the string specified in the *Name* parameter is a valid iSCSI name ([RFC3720] section 3.2.6), and if it is not, the server SHOULD return ERROR_INVALID_DATA.
- Checks whether the specified name is being used by an existing target, and if it is, the server SHOULD return ERROR_FILE_EXISTS.
- Sets the value of the **Name** data element (section 3.1.1.7) of the target with the specified name.

3.21.4.6 [propget]Description (Opnum 8)

The **[propget]Description** method returns the description of the target.

```
[propget] HRESULT Description(
        [out] [retval] BSTR * pDescription);
```

pDescription: A pointer to a string that returns the description of the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Description** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **Description** data element (section 3.1.1.7) of the target by using the *pDescription* parameter.

3.21.4.7 [propput]Description (Opnum 9)

The **[propput]Description** method sets the description of the target.

```
[propput] HRESULT Description(
        [in] BSTR Description);
```

Description: The description of the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]Description** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server sets the value of the **Description** data element (section 3.1.1.7) of the target from the *Description* parameter.

3.21.4.8 [propget]TargetIQN (Opnum 10)

The **[propget]TargetIQN** method returns the IQN of the target.

```
[propget] HRESULT TargetIQN(
        [out] [retval] BSTR * pTargetIQN);
```

pTargetIQN: A pointer to a string that returns the IQN of the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]TargetIQN** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **TargetIQN** data element (section 3.1.1.7) of the target by using the *pTargetIQN* parameter.

3.21.4.9 [propput]TargetIQN (Opnum 11)

The **[propput]TargetIQN** method sets the IQN of the target.

```
[propput] HRESULT TargetIQN(
        [in] BSTR TargetIQN);
```

TargetIQN: The IQN of the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.
E_INVALIDARG 0x80070057	One or more arguments are invalid.
ERROR_FILE_EXISTS 0x80070050	The supplied IQN is already in use by a target.
ERROR_INVALID_DATA 0x8007000d	The syntax of the supplied target IQN name is invalid.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]TargetIQN** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server

SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Checks whether the string specified in the *TargetIQN* parameter is a valid iSCSI name ([RFC3720] section 3.2.6), and if it is not, the server SHOULD return ERROR_INVALID_DATA.
- Sets the value of the **TargetIQN** data element (section 3.1.1.7) of the target from the *TargetIQN* parameter.

3.21.4.10 [propget]Status (Opnum 12)

The **[propget]Status** method returns the connected status of the target.

```
[propget] HRESULT Status(
        [out] [retval] HostStatus * pStatus);
```

pStatus: A pointer to a **HostStatus** enumeration value (section 2.2.3.4) that returns the connected status of the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Status** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Sets the **Status** data element to **HostStatusIdle** (section 2.2.3.4) if there are no active sessions, or to **HostStatusConnected** (section 2.2.3.4) if there are active sessions.
- Returns the value of the **Status** data element (section 3.1.1.7) of the target by using the *pStatus* parameter.

3.21.4.11 [propput]Status (Opnum 13)

The **[proput]Status** method MAY set the connected status of the target.<20>

```
[propput] HRESULT Status(
        [in] HostStatus eStatus);
```

eStatus: A **HostStatus** enumeration value (section 2.2.3.4) that specifies the connected status of the target.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.21.4.12 [propget]LastLogIn (Opnum 14)

The **[propget]LastLogIn** method returns the time stamp of the last login on the target.

```
[propget] HRESULT LastLogIn(
        [out] [retval] hyper * pTimeStamp);
```

pTimeStamp: A pointer to a signed value that returns the time stamp of the last login on the target by an iSCSI initiator. This value SHOULD be interpreted as a **FILETIME** structure (section 2.2).

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]LastLogIn** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

Otherwise, the server returns the value of the **LastLogIn** data element (section 3.1.1.7) of the target by using the *pTimeStamp* parameter.

The **LastLogIn** data element SHOULD be updated every time a session that addresses the target is logged in.

3.21.4.13 [propput]LastLogIn (Opnum 15)

The **[propput]LastLogIn** method MAY set the time stamp of the last login on the target.<21>

```
[propput] HRESULT LastLogIn(
        [in] hyper hyTimeStamp);
```

hyTimeStamp: The time stamp of the last login on the target by an iSCSI initiator. The value of this parameter SHOULD be interpreted as a **FILETIME** structure (section 2.2).

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.21.4.14 [propget]EnableCHAP (Opnum 16)

The **[propget]EnableCHAP** method returns whether CHAP authentication is required for an initiator to connect to the target.

```
[propget] HRESULT EnableCHAP(
        [out] [retval] boolean * pEnable);
```

pEnable: A pointer to a Boolean value that returns whether CHAP authentication is required on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]EnableCHAP** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **EnableCHAP** data element (section 3.1.1.7) of the target by using the *pEnable* parameter.

3.21.4.15 [propput]EnableCHAP (Opnum 17)

The **[propput]EnableCHAP** method sets whether CHAP authentication is required for an initiator to connect to the target.

```
[propput] HRESULT EnableCHAP(
        [in] boolean bEnable);
```

bEnable: A Boolean value that specifies whether CHAP authentication is required on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]EnableCHAP** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Sets the value of the **EnableCHAP** data element (section 3.1.1.7) of the target from the *bEnable* parameter.
- When the **EnableCHAP** data element is set to true, the **CHAPUserName** and **CHAPSecret** data elements (section 3.1.1.7) SHOULD be evaluated and used for subsequent session authentication.

3.21.4.16 [propget]CHAPUserName (Opnum 18)

The **[propget]CHAPUserName** method returns the user name for CHAP authentication on the target.

```
[propget] HRESULT CHAPUserName(
        [out] [retval] BSTR * pUserName);
```

pUserName: A pointer to a string that returns the user name for CHAP authentication on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]CHAPUserName** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server

SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **CHAPUserName** data element (section 3.1.1.7) of the target by using the *pUserName* parameter.

3.21.4.17 [propput]CHAPUserName (Opnum 19)

The **[propput]CHAPUserName** method sets the user name for CHAP authentication on the target.

```
[propput] HRESULT CHAPUserName(
        [in] BSTR UserName);
```

UserName: The user name for CHAP authentication on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]CHAPUserName** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server sets the value of the **CHAPUserName** data element (section 3.1.1.7) of the target from the *UserName* parameter.

The **CHAPUserName** data element is applicable and SHOULD be used for session authentication if the **EnableCHAP** data element (section 3.1.1.7) is set to true.

3.21.4.18 [propget]CHAPSecret (Opnum 20)

The **[propget]CHAPSecret** method returns the CHAP secret for CHAP authentication on the target.

```
[propget] HRESULT CHAPSecret(
        [out] [retval] BSTR * pSecret);
```

pSecret: A pointer to a string that returns the CHAP secret for CHAP authentication on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]CHAPSecret** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **CHAPSecret** data element (section 3.1.1.7) of the target by using the *pSecret* parameter.

3.21.4.19 [propput]CHAPSecret (Opnum 21)

The **[propput]CHAPSecret** method sets the CHAP secret for CHAP authentication on the target.

```
[propput] HRESULT CHAPSecret(
        [in] BSTR Secret);
```

Secret: The CHAP secret for CHAP authentication on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

Return value/code	Description
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]CHAPSecret** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server sets the value of the **CHAPSecret** data element (section 3.1.1.7) of the target from the *Secret* parameter.

The **CHAPSecret** data element is applicable and SHOULD be used for session authentication if the **EnableCHAP** data element (section 3.1.1.7) is set to true.

3.21.4.20 [propget]EnableReverseCHAP (Opnum 22)

The **[propget]EnableReverseCHAP** method returns whether CHAP mutual authentication is required for an initiator to connect to the target.

```
[propget] HRESULT EnableReverseCHAP(
        [out] [retval] boolean * pEnable);
```

pEnable: A pointer to a Boolean value that returns whether CHAP mutual authentication is required for an initiator to connect to the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]EnableReverseCHAP** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **EnableReverseCHAP** data element (section 3.1.1.7) of the target by using the *pEnable* parameter.

3.21.4.21 [propput]EnableReverseCHAP (Opnum 23)

The **[propput]EnableReverseCHAP** method sets whether CHAP mutual authentication is required for an initiator to connect to the target.

```
[propput] HRESULT EnableReverseCHAP(
        [in] boolean bEnable);
```

bEnable: A Boolean value that specifies whether CHAP mutual authentication is required for an initiator to connect to the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]EnableReverseCHAP** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Sets the value of the **EnableReverseCHAP** data element (section 3.1.1.7) of the target from the *bEnable* parameter.
- When the **EnableReverseCHAP** data element is set to true, the **ReverseCHAPUserName** and **ReverseCHAPSecret** data elements (section 3.1.1.7) SHOULD be evaluated and used for subsequent session authentication.

3.21.4.22 [propget]ReverseCHAPUserName (Opnum 24)

The **[propget]ReverseCHAPUserName** method returns the user name for CHAP mutual authentication on the target.

```
[propget] HRESULT ReverseCHAPUserName(
        [out] [retval] BSTR * pUserName);
```

pUserName: A pointer to a string that returns the user name for CHAP mutual authentication on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]ReverseCHAPUserName** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **ReverseCHAPUserName** data element (section 3.1.1.7) of the target by using the *pUserName* parameter.

3.21.4.23 [propput]ReverseCHAPUserName (Opnum 25)

The **[propput]ReverseCHAPUserName** method sets the user name for CHAP mutual authentication on the target.

```
[propput] HRESULT ReverseCHAPUserName(
        [in] BSTR UserName);
```

UserName: The user name for CHAP mutual authentication on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]ReverseCHAPUserName** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server sets the value of the **ReverseCHAPUserName** data element (section 3.1.1.7) of the target from the *UserName* parameter.

The **ReverseCHAPUserName** data element is applicable and SHOULD be used for session authentication if the **EnableReverseCHAP** data element (section 3.1.1.7) is set to true.

3.21.4.24 [propget]ReverseCHAPSecret (Opnum 26)

The **[propget]ReverseCHAPSecret** method returns the CHAP secret for CHAP mutual authentication on the target.

```
[propget] HRESULT ReverseCHAPSecret(
        [out] [retval] BSTR * pSecret);
```

pSecret: A pointer to a string that returns the CHAP secret for CHAP mutual authentication on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.

Return value/code	Description
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]ReverseCHAPSecret** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **ReverseCHAPSecret** data element (section 3.1.1.7) of the target by using the *pSecret* parameter.

3.21.4.25 [propput]ReverseCHAPSecret (Opnum 27)

The **[propput]ReverseCHAPSecret** method sets the CHAP secret for CHAP mutual authentication on the target.

```
[propput] HRESULT ReverseCHAPSecret(
        [in] BSTR Secret);
```

Secret: The CHAP secret for CHAP mutual authentication on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]ReverseCHAPSecret** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server sets the value of the **ReverseCHAPSecret** data element (section 3.1.1.7) of the target from the *Secret* parameter.

The **ReverseCHAPSecret** data element is applicable and SHOULD be used for session authentication if the **EnableReverseCHAP** data element (section 3.1.1.7) is set to true.

3.21.4.26 [propget]TargetMaxRecvDataSegmentLength (Opnum 28)

The **[propget]TargetMaxRecvDataSegmentLength** method returns the maximum receive-data segment length for the target.

```
[propget] HRESULT TargetMaxRecvDataSegmentLength(
        [out] [retval] unsigned long * pLength);
```

pLength: A pointer to a value that returns the maximum receive-data segment length for the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]TargetMaxRecvDataSegmentLength** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **TargetMaxRecvDataSegmentLength** data element (section 3.1.1.7) of the target by using the *pLength* parameter.<22>

3.21.4.27 [propput]TargetMaxRecvDataSegmentLength (Opnum 29)

The **[propput]TargetMaxRecvDataSegmentLength** method sets the maximum receive-data segment length for the target.

```
[propput] HRESULT TargetMaxRecvDataSegmentLength(
        [in] unsigned long nLength);
```

nLength: The maximum receive-data segment length for the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.
E_INVALIDARG 0x80070057	The supplied configuration option is not valid or compatible with implementation limits or constrains.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]TargetMaxRecvDataSegmentLength** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Checks whether the specified length value is compatible with implementation limits.<23> If the specified length is not compatible, the server SHOULD return E_INVALIDARG
- Sets the value of the **TargetMaxRecvDataSegmentLength** data element (section 3.1.1.7) of the target from the *nLength* parameter.

3.21.4.28 [propget]TargetFirstBurstLength (Opnum 30)

The **[propget]TargetFirstBurstLength** method returns the first burst length for the target.

```
[propget] HRESULT TargetFirstBurstLength(
        [out] [retval] unsigned long * pLength);
```

pLength: A pointer to a value that returns the first burst length for the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]TargetFirstBurstLength** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **TargetFirstBurstLength** data element (section 3.1.1.7) of the target by using the *pLength* parameter.<24>

3.21.4.29 [propput]TargetFirstBurstLength (Opnum 31)

The **[propput]TargetFirstBurstLength** sets the first burst length for the target.

```
[propput] HRESULT TargetFirstBurstLength(
        [in] unsigned long nLength);
```

nLength: The first burst length for the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

Return value/code	Description
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.
E_INVALIDARG 0x80070057	The supplied configuration option is not valid or compatible with implementation limits or constrains.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]TargetFirstBurstLength** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Checks whether the specified length value is compatible with implementation limits.<25> If the specified length is not compatible, the server SHOULD return E_INVALIDARG.
- Sets the value of the **TargetFirstBurstLength** data element (section 3.1.1.7) of the target from the *nLength* parameter.

3.21.4.30 [propget]TargetMaxBurstLength (Opnum 32)

The **[propget]TargetMaxBurstLength** method returns the maximum burst length for the target.

```
[propget] HRESULT TargetMaxBurstLength(
        [out] [retval] unsigned long * pLength);
```

pLength: A pointer to a value that returns the maximum burst length for the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]TargetMaxBurstLength** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **TargetMaxBurstLength** data element (section 3.1.1.7) of the target by using the *pLength* parameter.<26>

3.21.4.31 [propput]TargetMaxBurstLength (Opnum 33)

The **[propput]TargetMaxBurstLength** method sets the maximum burst length for the target.

```
[propput] HRESULT TargetMaxBurstLength(
        [in] unsigned long nLength);
```

nLength: The maximum burst length for the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.
E_INVALIDARG 0x80070057	The supplied configuration option is not valid or compatible with implementation limits or constrains.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]TargetMaxBurstLength** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Checks whether the specified length value is compatible with implementation limits.<27> If the specified length is not compatible, the server SHOULD return E_INVALIDARG.

- Sets the value of the **TargetMaxBurstLength** data element (section 3.1.1.7) of the target from the *nLength* parameter.

3.21.4.32 [propget]NumRecvBuffers (Opnum 34)

The **[propget]NumRecvBuffers** method returns the number of receive buffers for the target.

```
[propget] HRESULT NumRecvBuffers(
        [out] [retval] unsigned long * pNumBuffers);
```

pNumBuffers: A pointer to a value that returns the number of receive buffers for the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]NumRecvBuffers** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **NumRecvBuffers** data element (section 3.1.1.7) of the target by using the *pNumBuffers* parameter.<28>

3.21.4.33 [propput]NumRecvBuffers (Opnum 35)

The **[propput]NumRecvBuffers** method sets the number of receive buffers for the target.

```
[propput] HRESULT NumRecvBuffers(
        [in] unsigned long nNumBuffers);
```

nNumBuffers: The number of receive buffers for the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.
E_INVALIDARG 0x80070057	The supplied configuration option is not valid or compatible with implementation limits or constrains.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]NumRecvBuffers** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Checks whether the specified length value is compatible with implementation limits.<29> If the specified number of receive buffers is not compatible, the server SHOULD return E_INVALIDARG.
- Sets the value of the **NumRecvBuffers** data element (section 3.1.1.7) of the target from the *nNumBuffers* parameter.

3.21.4.34 [propget]EnforceIdleTimeoutDetection (Opnum 36)

The **[propget]EnforceIdleTimeoutDetection** method returns whether idle timeout detection is enforced on the target.

```
[propget] HRESULT EnforceIdleTimeoutDetection(
        [out] [retval] boolean * pEnable);
```

pEnable: A pointer to a Boolean value that returns whether idle timeout detection is enforced on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]EnforceIdleTimeoutDetection** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the value of the **EnforceIdleTimeoutDetection** data element (section 3.1.1.7) of the target by using the *pEnable* parameter.<30>

3.21.4.35 [propput]EnforceIdleTimeoutDetection (Opnum 37)

The **[propput]EnforceIdleTimeoutDetection** method sets whether idle timeout detection is enforced on the target.

```
[propput] HRESULT EnforceIdleTimeoutDetection(
          [in] boolean bEnable);
```

bEnable: A Boolean value that specifies whether idle timeout detection is enforced on the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]EnforceIdleTimeoutDetection** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server sets the value of the **EnforceIdleTimeoutDetection** data element (section 3.1.1.7) of the target from the *bEnable* parameter.

3.21.4.36 [propget]ListenAllNetworkAdapters (Opnum 38)

The **[propget]ListenAllNetworkAdapters** method MAY return whether the target listens on all network adapters.<31>

```
[propget] HRESULT ListenAllNetworkAdapters(
        [out] [retval] boolean * pListenAll);
```

pListenAll: A pointer to a Boolean value that returns whether the target listens on all network adapters.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.21.4.37 [propput]ListenAllNetworkAdapters (Opnum 39)

The **[propput]ListenAllNetworkAdapters** method MAY set whether the target listens on all network adapters.<32>

```
[propput] HRESULT ListenAllNetworkAdapters(
        [in] boolean bListenAll);
```

bListenAll: A Boolean value that specifies whether the target listens on all network adapters.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.21.4.38 [propget]Processor (Opnum 40)

The **[propget]Processor** method MAY return the physical processor that is designated as the processing resource for requests directed to the target.<33>

```
[propget] HRESULT Processor(
        [out] [retval] unsigned long * pProcessor);
```

pProcessor: A pointer to a 32-bit bitmask that returns the processor for the target, expressed as a bit setting.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.21.4.39 [propput]Processor (Opnum 41)

The **[propput]Processor** method MAY set the physical processor that is designated as the processing resource for requests directed to the target.<34>

```
[propput] HRESULT Processor(
        [in] unsigned long nProcessor);
```

nProcessor: A 32-bit bitmask that specifies the processor for the target, expressed as a bit setting.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.21.4.40 EnumIDs (Opnum 42)

The **EnumIDs** method returns an enumeration of identification methods for the iSCSI initiators that are configured for the target.

```
HRESULT EnumIDs(
        [out] IEnumIDMethod ** ppEnumIDMethod);
```

ppEnumIDMethod: A pointer to an **IEnumIDMethod** interface (section 3.9) pointer that returns the identification method enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumIDs** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Initializes and populates a collection of identification information in an **IDMethodList** data object (section 3.1.1.7). Each element SHOULD be an instance of **ID** structure (section 2.2.4.7).
- Creates an **IEnumIDMethod** interface object and initializes it with the collection of identification information.
- Returns a pointer to the **IEnumIDMethod** interface pointer by using the *ppEnumIDMethod* parameter.

3.21.4.41 AddID (Opnum 43)

The **AddID** method adds a specified identification method for an iSCSI initiator to the target.

```
HRESULT AddID(
        [in] ID * pIDMethod);
```

pIDMethod: A pointer to an **ID** structure (section 2.2.4.7) that specifies the identification method for an iSCSI initiator, which is to be added to the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.
E_INVALIDARG 0x80070057	The supplied initiator identification method is in an invalid format.
ERROR_INVALID_DATA 0x8007000d	The syntax of the supplied identification method is invalid.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **AddID** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Checks the format and syntax of the specified identification method, and returns an appropriate error if it is not valid.
- Checks whether the specified identification method is already registered for the target. If it is, the server SHOULD return S_OK.
- Adds the specified identification method information to the **IDMethodList** data object (section 3.1.1.7).

3.21.4.42 RemoveID (Opnum 44)

The **RemoveID** method removes a specified identification method for an iSCSI initiator from the target.

```
HRESULT RemoveID(
        [in] ID * pIDMethod);
```

pIDMethod: A pointer to an **ID** structure (section 2.2.4.7) that specifies the identification method for an iSCSI initiator, which is to be removed from the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **RemoveID** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Finds the specified identification method for the target. If it is not found, the server SHOULD return S_OK.
- Removes the specified identification method information from the **IDMethodList** data object (section 3.1.1.7).
- For the identification method that was removed, find the sessions that were authenticated using that method and disconnect them from the target.

3.21.4.43 RemoveAllIDs (Opnum 45)

The **RemoveAllIDs** method removes all identification methods from the target.

```
HRESULT RemoveAllIDs();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

Return value/code	Description
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **RemoveAllIDs** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Removes all identification method information from the **IDMethodList** data object (section 3.1.1.7).
- For each identification method that was removed, find the sessions that were authenticated using that method, and disconnect them from the target.

3.21.4.44 EnumWTDiskLunMappings (Opnum 46)

The **EnumWTDiskLunMappings** method returns an enumeration of the mappings of LUN numbers [SAM-3] to virtual disks that are assigned to the target.

```
HRESULT EnumWTDiskLunMappings(
        [out] IEnumWTDiskLunMapping ** ppEnumWTDiskLunMapping);
```

ppEnumWTDiskLunMapping: A pointer to an **IEnumWTDiskLunMapping** interface (section 3.20) pointer that returns the enumeration of the mappings of LUN numbers to virtual disks. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation succeeded, but the enumerator might not have been created or initialized.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumWTDiskLunMappings** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Initializes and populates a collection of LUN mapping information in a **WTDiskLunMappingsList** data object (section 3.1.1.7). Each element SHOULD be an instance of **LUN_MAPPING** structure (section 2.2.4.10).
- Creates an **IEnumWTDiskLunMapping** interface object and initializes it with the collection of LUN mapping information.
- Returns a pointer to the **IEnumWTDiskLunMapping** interface pointer by using the *ppEnumWTDiskLunMapping* parameter.

3.21.4.45 AddWTDisk (Opnum 47)

The **AddWTDisk** method adds a specified virtual disk assignment to the target.

```
HRESULT AddWTDisk(
        [in] unsigned long nWTD);
```

nWTD: The virtual disk index to add.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online or the device could not be assigned to existing initiators' sessions.
ERROR_FILE_NOT_FOUND 0x80070002	The specified LUN identifier is not found.
ERROR_DELETE_PENDING 0x8007012f	The specified LUN identifier is in a state that is incompatible with the operation requested.
E_INVALIDARG	In an HA cluster configuration, the resource group assignment is

Return value/code	Description
0x80070057	incompatible with the operation requested.
ERROR_HANDLE_EOF 0x80070026	The specified LUN identifier could not be added because an internal limit has been reached.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **AddWTDisk** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the following:

- The **ResourceState** data element (section 3.1.1.7) to determine whether the target is online; if it is not, the server SHOULD return ERROR_INVALID_FUNCTION.
- Whether the disk belongs to the same resource group as the target; if it does not, the server SHOULD return E_INVALIDARG.

The server SHOULD process this method as follows:

- Checks whether the disk is already assigned to the target; if so, the server SHOULD return S_OK.
- Checks whether the disk exists and whether it can be assigned to the target; if not, the server SHOULD return an appropriate error. <35>
- Updates the **WTDiskLunMappingsList** data object (section 3.1.1.7) with the new LUN mapping information.
- Sends notifications of the change in LUN information to iSCSI initiators that are connected to the target.
- Sends notifications of the change in LUN information to registered ISTM clients by using the **IStatusNotify** interface (section 3.42).

The newly added virtual disk gets exposed to the initiator with a server-assigned LUN number. The assignment starts from zero and uses the first available index in the **WTDiskLunMappingsList** data object. The assignment can be changed as specified in section 3.21.4.48.

3.21.4.46 RemoveWTDisk (Opnum 48)

The **RemoveWTDisk** method removes a specified virtual disk assignment from the target.

```
HRESULT RemoveWTDisk(
        [in] unsigned long nWTD);
```

nWTD: The disk index to remove.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation succeeded or the specified LUN index is not associated with the target.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.
ERROR_FILE_NOT_FOUND 0x80070002	The specified LUN identifier is not currently assigned to the target.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **RemoveWTDisk** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Checks whether the virtual disk is assigned to the target; if not, the server SHOULD return ERROR_FILE_NOT_FOUND.
- Updates the **WTDiskLunMappingsList** data object (section 3.1.1.7) by removing the LUN mapping information for the specified virtual disk.
- Sends notifications of the change in LUN information to iSCSI initiators that are connected to the target.
- Sends notifications of the change in LUN information to registered ISTM clients by using the **IStatusNotify** interface (section 3.42).

3.21.4.47 RemoveAllWTDisks (Opnum 49)

The **RemoveAllWTDisks** method removes all virtual disk assignments from the target.

```
HRESULT RemoveAllWTDisks();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

Return value/code	Description
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Checks whether any disks are assigned to the target; if not, the server SHOULD return S_OK.
- Updates the **WTDiskLunMappingsList** data object (section 3.1.1.7) by removing the LUN mapping information for all disks.
- Sends notifications of the change in LUN information to iSCSI initiators that are connected to the target.
- Sends notifications of the change in LUN information to registered ISTM clients by using the **IStatusNotify** interface (section 3.42).

3.21.4.48 SetWTDiskLunMapping (Opnum 50)

The **SetWTDiskLunMapping** method sets the LUN mapping for the specified disk.

```
HRESULT SetWTDiskLunMapping(
        [in] unsigned long nWTD,
        [in] unsigned long nNewLun);
```

nWTD: The virtual disk index.

nNewLun: The LUN number, as seen at the **SCSI** level by iSCSI initiators [SAM-3].

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online.
E_INVALIDARG	The specified LUN number is outside of the admissible range.

Return value/code	Description
0x80070057	
ERROR_FILE_EXISTS 0x80070050	The specified LUN number is already assigned.
ERROR_FILE_NOT_FOUND 0x80070002	The specified LUN device is not found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **SetWTDiskLUNMapping** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Validates the LUN index and ensures that the LUN index is not already assigned. If the LUN index is not valid, the server SHOULD return an appropriate error. <36>
- Validates the virtual disk index and whether it is assigned to the target; if not, the server SHOULD return an appropriate error.
- Maps the specified LUN to the specified virtual disk.
- Updates the **WTDiskLunMappingsList** data object (section 3.1.1.7) with the changed LUN mapping information.
- Sends notifications of the change in LUN information to iSCSI initiators that are connected to the target.
- Sends notifications of the change in LUN information to registered ISTM clients by using the **IStatusNotify** interface (section 3.42).

3.21.4.49 EnumPortals (Opnum 51)

The **EnumPortals** method returns an enumeration of the portals [RFC3720] that are configured to service iSCSI initiator requests for the target.

```
HRESULT EnumPortals(
        [out] IEnumPortal ** ppEnumPortal);
```

ppEnumPortal: A pointer to an **IEnumPortal** interface (section 3.11) pointer that returns the enumeration of portal information. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumPortals** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Creates a collection of information for all portals on the iSCSI Software Target computer, with the elements of the collection defined according to the abstract data model (section 3.1.1.3.5) that is relevant to the **PORTAL** structure (section 2.2.4.13).
- Creates an **IEnumPortal** interface object and MAY associate with it the collection of portal information.<37>
- Returns a pointer to the **IEnumPortal** interface pointer by using the *ppEnumPortal* parameter.

3.21.4.50 AddPortal (Opnum 52)

The **AddPortal** method MAY add a specified portal to the target.<38>

```
HRESULT AddPortal(
        [in] BSTR IpAddress);
```

IpAddress: A string representation of the IP address of the portal to be added.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.21.4.51 RemovePortal (Opnum 53)

The **RemovePortal** method MAY remove a specified portal from the target.<39>

```
HRESULT RemovePortal(
        [in] BSTR IpAddress);
```

IpAddress: A string representation of the IP address of the portal to be removed.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation succeeded but had no side effects.

3.21.4.52 RemoveAllPortals (Opnum 54)

The **RemoveAllPortals** method MAY remove all portals from the target.<40>

```
HRESULT RemoveAllPortals();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation succeeded but had no side effects.

3.21.4.53 SetCHAPInfoByBlob (Opnum 55)

The **SetCHAPInfoByBlob** method MAY set CHAP information for the target by using the data in a specified encrypted BLOB.<41>

```
HRESULT SetCHAPInfoByBlob(
        [in] unsigned long nBlobSize,
        [in] [size_is(nBlobSize)] char Blob[]);
```

nBlobSize: The size of the encrypted BLOB.

Blob: The encrypted data.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.21.4.54 Save (Opnum 56)

The **Save** method saves target properties to persistent storage.

```
HRESULT Save();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.

When the **Save** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

The server SHOULD save to persistent storage all relevant target elements specified in the target management data model (section 3.1.1.7) and return S_OK.

3.21.4.55 ValidateChapSecret (Opnum 57)

The **ValidateCHAPSecret** method validates the specified CHAP secret.

```
HRESULT ValidateChapSecret(
        [in] BSTR Secret);
```

Secret: The CHAP secret to be validated.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.
ERROR_BAD_LENGTH 0x80070018	The supplied secret does not satisfy the length requirement.
ERROR_DUP_NAME 0x80070034	The supplied secret does not satisfy the uniqueness requirement.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **ValidateChapSecret** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD validate the structure of the CHAP secret by executing an implementation-specific validation policy that includes entropy of the secret and minimum length requirements.<42>

3.21.4.56 ValidateReverseChapSecret (Opnum 58)

The **ValidateReverseChapSecret** method validates the specified mutual CHAP secret.

```
HRESULT ValidateReverseChapSecret(
        [in] BSTR Secret);
```

Secret: The mutual CHAP secret to be validated.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.

Return value/code	Description
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.
ERROR_BAD_LENGTH 0x80070018	The supplied secret does not satisfy length requirements.
ERROR_DUP_NAME 0x80070034	The supplied secret does not satisfy uniqueness requirements.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **ValidateReverseChapSecret** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

The server SHOULD validate the structure of the mutual CHAP secret by executing an implementation-specific validation policy that includes entropy of the secret and minimum length requirements.<43>

3.21.4.57 CheckTargetRedirect (Opnum 59)

The **CheckTargetRedirect** method MAY check whether redirection of the target is required.<44>

```
HRESULT CheckTargetRedirect(
        [in] BSTR LocalConnectAddr,
        [out] BSTR * pRedirectAddr);
```

LocalConnectAddr: The local address that the initiator has connected to.

pRedirectAddr: A pointer to a string that MAY return the redirection address and port.<45>

Return Values:

The following value is possible:

Return value/code	Description
S_FALSE 0x00000001	Redirection of the target is not required.

3.21.5 Timer Events

None.

3.21.6 Other Local Events

None.

3.22 IHost Client Details

The **IHost** interface (section 3.21) is used to manage a particular iSCSI target.

3.22.1 Abstract Data Model

The client data model is specified in section 3.2.1.

3.22.2 Timers

None.

3.22.3 Initialization

Client initialization is specified in section 3.2.3.

3.22.4 Message Processing Events and Sequencing Rules

This section specifies client processing of error returns by methods of the **IHost** interface (section 3.21).

3.22.4.1 AddWTDisk (Opnum 47)

Processing of the following error return SHOULD be performed by client callers of the **AddWTDisk** method (section 3.21.4.45):

ERROR_HANDLE_EOF

A return value of ERROR_HANDLE_EOF indicates that the specified virtual disk cannot be added to the target, because an implementation-defined limit of LUNs per target would be exceeded.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the specified virtual disk cannot be added to the target.<46>

3.22.5 Timer Events

None.

3.22.6 Other Local Events

None.

3.23 IHost2 Server Details

The **IHost2** interface extends the **IHost** interface (section 3.21) with support for HA clusters and non-persistent iSCSI targets.

3.23.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the target management data model (section 3.1.1.7).

3.23.2 Timers

None.

3.23.3 Initialization

Target manager initialization is specified in section 3.1.3.

3.23.4 Message Processing Events and Sequencing Rules

The **IHost2** interface includes the following methods:

Method	Description
[propget]ResourceName	Returns the name of the cluster resource that represents the target. Opnum: 60
[propput]ResourceName	Sets the name of the cluster resource that represents the target. Opnum: 61
[propget]ResourceGroupName	Returns the name of the resource group that the target belongs to. Opnum: 62
[propput]ResourceGroupName	Sets the name of the resource group that the target belongs to. Opnum: 63
SaveToStream	Saves target properties to a specified stream. Opnum: 64
EnumSessions	Returns an enumeration of sessions on the target. Opnum: 65

3.23.4.1 [propget]ResourceName (Opnum 60)

The **[propget]ResourceName** method returns the name of the cluster resource that represents the target.

```
[propget] HRESULT ResourceName(
        [out] [retval] BSTR * pResourceName);
```

pResourceName: A pointer to a string that returns the name of the cluster resource that represents the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

Return value/code	Description
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	One or more arguments are invalid.

When the **[propget]ResourceName** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Checks the *pResourceName* parameter; if it is invalid or NULL, the server SHOULD return E_INVALIDARG.
- Returns the **ResourceName** data element (section 3.1.1.7) by using the *pResourceName* parameter.

3.23.4.2 [propput]ResourceName (Opnum 61)

The **[propput]ResourceName** method sets the name of the cluster resource that represents the target.

```
[propput] HRESULT ResourceName(
        [in] BSTR ResourceName);
```

ResourceName: The name of the cluster resource that represents the target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	One or more arguments are invalid.

When the **[propput]ResourceName** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Checks the *ResourceName* parameter; if it is invalid, or its length is zero, the server SHOULD return E_INVALIDARG.
- Sets the **ResourceName** data element (section 3.1.1.7) from the *ResourceName* parameter.

3.23.4.3 [propget]ResourceGroupName (Opnum 62)

The **[propget]ResourceGroupName** method returns the name of the resource group that the target belongs to.

```
[propget] HRESULT ResourceGroupName(
        [out] [retval] BSTR * pResourceGroup);
```

pResourceGroup: A pointer to a string that returns the resource group that the target belongs to.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	One or more arguments are invalid.

When the **[propget]ResourceGroupName** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Checks the *pResourceGroupName* parameter; if it is invalid or NULL, the server SHOULD return E_INVALIDARG.
- In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server returns the **ResourceGroupName** data element (section 3.1.1.7) by using the *pResourceName* parameter.

- In a non-HA cluster configuration, the server SHOULD return an empty string by using the *pResourceName* parameter.

3.23.4.4 [propput]ResourceGroupName (Opnum 63)

The **[propput]ResourceGroupName** method MAY set the name of the resource group that the target belongs to.<47>

```
[propput] HRESULT ResourceGroupName(
        [in] BSTR ResourceGroup);
```

ResourceGroup: The name of the resource group that the target belongs to.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.23.4.5 SaveToStream (Opnum 64)

The **SaveToStream** method MAY save target properties to a specified stream.<48>

```
HRESULT SaveToStream(
        [in] IUnknown * pStream);
```

pStream: A pointer to an **IUnknown** interface [MS-DCOM] object.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	Access is denied. The operation is not allowed from a remote client.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.

When the **SaveToStream** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

The server SHOULD perform any transport and implementation-specific checks that determine the locality of the interface pointer specified by the *pStream* parameter. For a remote caller, the server SHOULD return E_ACCESSDENIED.

Additional processing by this method for a local caller is not required for protocol compliance, and its implementation MAY be omitted.<49>

3.23.4.6 EnumSessions (Opnum 65)

The **EnumSessions** method returns an enumeration of sessions on the target.

```
HRESULT EnumSessions(
       [out] IEnumSession2 ** ppEnumSession);
```

ppEnumSession: A pointer to an **IEnumSession2** interface (section 3.14) pointer that returns the session enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.
E_POINTER 0x80004003	Invalid pointer.

When the **EnumSessions** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.7) to determine whether the target is online, and if the target is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Creates a collection of information for all sessions on the iSCSI Software Target computer, with the elements of the collection defined according to the abstract data model (section 3.1.1.2.1) that is relevant to the **SESSION2** structure (section 2.2.4.16).

- Creates an **IEnumSession2** interface object and initializes it with the collection of session information.
- Returns the **IEnumSession2** interface pointer by using the *ppEnumSession* parameter.

3.23.5 Timer Events

None.

3.23.6 Other Local Events

None.

3.24 IHost3 Server Details

The **IHost3** interface extends the **IHost2** interface (section 3.23) with support for HA clusters.

3.24.1 Abstract Data Model

The **IHost3** interface provides an abstraction of a cluster resource for a target in an HA cluster environment. A cluster resource can be in one of several states, as listed in section 3.1.1.7. Methods of this interface allow the transition to the **ClusterResourceOnline** or **ClusterResourceOffline** state ([MS-CMRP] section 3.1.4.1.13) from any other state. The act of actually reaching the desired final state is an asynchronous operation. Intermediate states are the result and function of the HA cluster implementation; they can be queried, but they cannot be explicitly retrieved or set.

For more details concerning the abstract data organization for this interface, see the target management data model (section 3.1.1.7).

3.24.2 Timers

None.

3.24.3 Initialization

Target manager initialization is specified in section 3.1.3.

3.24.4 Message Processing Events and Sequencing Rules

The **IHost3** interface includes the following methods:

Method	Description
[propget]ResourceState	Returns the cluster state of the resource in an HA cluster environment. Opnum: 66
[propput]ResourceState	Sets the cluster state of the resource in an HA cluster environment. Opnum: 67

3.24.4.1 [propget]ResourceState (Opnum 66)

The **[propget]ResourceState** method returns the cluster state of the resource in an HA cluster environment.

```
[propget] HRESULT ResourceState(
        [out] [retval] long * ResourceState);
```

ResourceState: A pointer to a value that returns the cluster state of the resource.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

The resource state in an HA cluster environment is maintained in the **ResourceState** data object (section 3.1.1.7).<50> If the iSCSI Software Target is not configured as part of an HA cluster, the cluster state SHOULD be **ClusterResourceStateUnknown** ([MS-CMRP] section 3.1.4.1.13).

3.24.4.2 [propput]ResourceState (Opnum 67)

The **[proput]ResourceState** method MAY set the cluster state of the resource in an HA cluster environment.<51>

```
[propput] HRESULT ResourceState(
        [in] long ResourceState);
```

ResourceState: A value that specifies the cluster state of the resource.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.24.5 Timer Events

None.

3.24.6 Other Local Events

None.

3.25 IHostMgr Server Details

The **IHostMgr** interface manages information about iSCSI targets that initiators can connect to.

3.25.1 Abstract Data Model

The **IHostMgr** interface provides the means to create, delete or retrieve an iSCSI target in the iSCSI Software Target.

For more details concerning the abstract data organization for this interface, see the target management data model (section 3.1.1.7).

3.25.2 Timers

None.

3.25.3 Initialization

Target manager initialization is specified in section 3.1.3.

3.25.4 Message Processing Events and Sequencing Rules

The **IHostMgr** interface includes the following methods:

Method	Description
EnumHosts	Returns an enumeration of the targets in the iSCSI Software Target. Opnum: 3
NewHost	Creates a new target with a specified name in the iSCSI Software Target. Opnum: 4
GetHost	Retrieves a target in the iSCSI Software Target by specifying the name of the host. Opnum: 5
GetHostByTargetIQN	Retrieves a target in the iSCSI Software Target by specifying the iSCSI qualified name (IQN) of the target. Opnum: 6
DeleteHost	Deletes a specified target from the iSCSI Software Target. Opnum: 7

3.25.4.1 EnumHosts (Opnum 3)

The **EnumHosts** method returns an enumeration of the targets in the iSCSI Software Target.

```
HRESULT EnumHosts(
        [out] IEnumHost ** ppEnumHost);
```

ppEnumHost: A pointer to an **IEnumHost** interface (section 3.8) pointer that returns the target enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumHosts** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates or updates a collection of **IHost** interface objects (section 3.21) that define all iSCSI targets that exist on the iSCSI Software Target, and saves the result in a **HostList** data object (section 3.1.1.7).
- Creates an **IEnumHost** interface object and initializes it with the collection in **HostList**.
- Returns a pointer to the **IEnumHost** interface by using the *ppEnumHost* parameter.

In an HA cluster environment, only targets with a **ResourceState** data element (section 3.1.1.7) value of **ClusterResourceOnline** ([MS-CMRP] section 3.1.4.1.13) SHOULD be returned by this method.

3.25.4.2 NewHost (Opnum 4)

The **NewHost** method creates a new target with a specified name in the iSCSI Software Target.

```
HRESULT NewHost(
        [in] BSTR HostName,
        [out] IHost ** ppNewHost);
```

HostName: The name of the target.

ppNewHost: A pointer to an **IHost** interface (section 3.21) pointer that returns the new target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.

Return value/code	Description
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	A supplied parameter is invalid.
ERROR_FILE_EXISTS 0x80070050	A target with the same name already exists.
ERROR_INVALID_DATA 0x8007000d	The target name is in an invalid format.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **NewHost** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Validates the input parameters, returning E_INVALIDARG if any of the input parameters are not valid.
- Creates an **IHost** interface object for a new target with the specified name.
- Adds the new **IHost** interface object to the collection of targets maintained in the **HostList** data object (section 3.1.1.7).
- Returns a pointer to the **IHost** interface object by using the *ppNewHost* parameter.

If the ISTM server is running in an HA cluster environment, this method SHOULD fail with E_INVALIDARG. The **NewHost2** method (section 3.27.4.1) SHOULD be used to accomplish this task in an HA cluster environment.

3.25.4.3 GetHost (Opnum 5)

The **GetHost** method retrieves a target in the iSCSI Software Target by specifying the name of the host.

```
HRESULT GetHost(
        [in] BSTR HostName,
        [out] IHost ** ppHost);
```

HostName: A string that contains the host name of the target to be retrieved.

ppHost: A pointer to an **IHost** interface (section 3.21) pointer that returns the requested target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	A supplied parameter is invalid.
ERROR_FILE_NOT_FOUND 0x80070002	The specified target name does not correspond to any existing target.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **GetHost** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Validates the input parameters, returning E_INVALIDARG if any of the input parameters are not valid.
- Finds the specified target in the collection maintained in the **HostList** data object (section 3.1.1.7).
- Returns the **IHost** interface pointer to the requested target object by using the *ppHost* parameter.

In an HA cluster environment, only targets with a **ResourceState** data element (section 3.1.1.7) value of **ClusterResourceOnline** or **ClusterResourceOffline** ([MS-CMRP] section 3.1.4.1.13) SHOULD be returned by this method.

3.25.4.4 GetHostByTargetIQN (Opnum 6)

The **GetHostByTargetIQN** method retrieves a target in the iSCSI Software Target by specifying the iSCSI qualified name (IQN) of the target.

```
HRESULT GetHostByTargetIQN(
        [in] BSTR TargetIQN,
        [out] IHost ** ppHost);
```

TargetIQN: A string that contains the IQN of the target to be retrieved.

ppHost: A pointer to an **IHost** interface (section 3.21) pointer that returns the requested target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	A supplied parameter is invalid.
ERROR_FILE_NOT_FOUND 0x80070002	The supplied IQN does not correspond to any existing target.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **GetHostByTargetIQN** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Validates the input parameters, returning E_INVALIDARG if any of the input parameters are not valid.
- Finds the specified target in the collection maintained in the **HostList** data object (section 3.1.1.7).
- Returns the **IHost** interface pointer to the requested target object by using the *ppHost* parameter.

In an HA cluster environment, only targets with a **ResourceState** data element (section 3.1.1.7) value of **ClusterResourceOnline** ([MS-CMRP] section 3.1.4.1.13) SHOULD be returned by this method.

3.25.4.5 DeleteHost (Opnum 7)

The **DeleteHost** method deletes a specified target from the iSCSI Software Target.

```
HRESULT DeleteHost(
        [in] BSTR HostName);
```

HostName: A string that contains the name of the target that is marked for deletion.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	A supplied parameter is invalid.
ERROR_FILE_NOT_FOUND 0x80070002	The specified target name does not correspond to any existing target.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **DeleteHost** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Validates the input parameters, returning E_INVALIDARG if any of the input parameters are not valid.
- Finds the specified target in the collection maintained in the **HostList** data object (section 3.1.1.7).
- Deletes the target from the **HostList** collection.

In an HA cluster environment, this method SHOULD delete targets that are in any of the cluster resource states specified for the **ResourceState** data object (section 3.1.1.7), including the **ClusterResourceOffline** state ([MS-CMRP] section 3.1.4.1.13).

3.25.5 Timer Events

None.

3.25.6 Other Local Events

None.

3.26 IHostMgr Client Details

The **IHostMgr** interface (section 3.25) is used to manage information about iSCSI targets that initiators can connect to.

3.26.1 Abstract Data Model

The client data model is specified in section 3.2.1.

3.26.2 Timers

None.

3.26.3 Initialization

Client initialization is specified in section 3.2.3.

3.26.4 Message Processing Events and Sequencing Rules

This section specifies client processing of error returns by methods of the **IHostMgr** interface (section 3.25).

3.26.4.1 NewHost (Opnum 4)

Processing of the following error returns SHOULD be performed by client callers of the **NewHost** method (section 3.25.4.2):

ERROR_FILE_EXISTS

A return value of ERROR_FILE_EXISTS indicates that an attempt has been made to create a new target with the name of a target that already exists.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the specified target name cannot be used.<52>

ERROR_INVALID_DATA

A return value of ERROR_INVALID_DATA indicates that the name specified for the new target is not valid. A possible cause is that the length of the name exceeds an implementation-defined limit.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the specified target name cannot be used.<53>

3.26.5 Timer Events

None.

3.26.6 Other Local Events

None.

3.27 IHostMgr2 Server Details

The **IHostMgr2** interface extends the **IHostMgr** interface (section 3.25) with support for non-persistent iSCSI targets.

3.27.1 Abstract Data Model

The **IHostMgr2** interface provides the means to create or delete an iSCSI target in the iSCSI Software Target when configured as part of an HA cluster. In an HA cluster, iSCSI targets are related to cluster resources, and some operations of this interface are expected to be performed by a resource in an HA cluster implementation.

For more details concerning the abstract data organization for this interface, see the target management data model (section 3.1.1.7).

3.27.2 Timers

None.

3.27.3 Initialization

Target manager initialization is specified in section 3.1.3.

3.27.4 Message Processing Events and Sequencing Rules

The **IHostMgr2** interface includes the following methods:

Method	Description
NewHost2	Creates a target in an HA cluster configuration in a specified resource group. Opnum: 8
NewHostNonPersistent	Creates a target in an HA cluster configuration in a specified resource group without saving the settings to persistent storage. Opnum: 9
OpenHost	Creates a target by loading its settings from a stream object. Opnum: 10
CloseHost	Closes a specified target. Opnum: 11

3.27.4.1 NewHost2 (Opnum 8)

The **NewHost2** method creates a target in an HA cluster configuration in a specified resource group.

```
HRESULT NewHost2(
        [in] BSTR HostName,
        [in] BSTR ResourceGroup,
        [out] IHost ** ppNewHost);
```

HostName: A string that contains the name of the target to be created.

ResourceGroup: A string that contains the name of the resource group in the HA cluster in which the target is to be created.

ppNewHost: A pointer to the **IHost** interface (section 3.21) pointer that returns the created target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

Return value/code	Description
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	A supplied parameter is invalid.
ERROR_FILE_EXISTS 0x80070050	A target with the same name already exists
ERROR_INVALID_DATA 0x8007000d	The target name is of an invalid format.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **NewHost2** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server processes this method as follows:

- Validates the *HostName* parameter, returning E_INVALIDARG if it is not valid. The name MUST be unique in the cluster.
- If a valid *ResourceGroup* parameter is specified, the server opens the specified resource group.
- If the *ResourceGroup* parameter is NULL or is an empty string, the server opens the first resource group owned by the server node or the first group in the cluster.
- Creates an **IHost** interface object for a new target with the specified name and brings it online.
- Adds the new **IHost** interface object to the collection of targets maintained in the **HostList** data object (section 3.1.1.7).
- Returns the **IHost** interface pointer to the newly created target object by using the *ppNewHost* parameter.

3.27.4.2 NewHostNonPersistent (Opnum 9)

The **NewHostNonPersistent** method creates a target in an HA cluster configuration in a specified resource group without saving the settings to persistent storage.

```
HRESULT NewHostNonPersistent(
        [in] BSTR HostName,
        [in] BSTR ResourceName,
        [out] IHost ** ppNewHost);
```

HostName: A string that contains the name of the target to be created.

ResourceName: A string that contains the name of the resource group in the HA cluster in which the target is to be created.

ppNewHost: A pointer to the **IHost** interface (section 3.21) pointer that returns the created target.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_FILE_EXISTS 0x80070050	The specified target name matches the name of an existing target.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server processes this method as follows:

- Checks whether the target already exists, returning ERROR_FILE_EXISTS if it does.
- Creates an **IHost** interface object for the target with the specified name in the specified resource group and brings it online.
- Adds the new **IHost** interface object to the collection of targets maintained in the **HostList** data object (section 3.1.1.7).
- Returns the **IHost** interface pointer to the newly created target object by using the *ppNewHost* parameter.

The target returned by this method SHOULD only have the **Guid**, **Name** and **ResourceName** properties (section 3.1.1.7) initialized. The **Guid** property is initialized to a random, globally-unique identifier, while the **Name** and **ResourceName** properties are initialized to the specified *HostName* and *ResourceName* parameters.

3.27.4.3 OpenHost (Opnum 10)

The **OpenHost** method MAY create a target by loading its settings from a stream object.<54>

```
HRESULT OpenHost(
        [in] IUnknown * pStream,
        [in] BSTR ResourceName,
        [out] IHost ** ppHost);
```

pStream: A pointer to an **IUnknown** interface [MS-DCOM] object that contains the target settings. The data contained or referenced as a payload in the stream is undefined, since this method does not process the data.

ResourceName: A string that contains the name of the cluster resource that represents the target.

ppHost: A pointer to an **IHost** interface (section 3.21) pointer that returns the target with the newly loaded configuration.

Return Values:

The following values are possible:

Return value/code	**Description**
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

When the **OpenHost** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD perform any transport or implementation-specific check that determines the locality of the interface pointer specified by the *pStream* parameter. For a remote caller, the server SHOULD return E_ACCESSDENIED.

Additional processing by this method for a local caller is not required for protocol compliance, and its implementation MAY be omitted.<55>

3.27.4.4 CloseHost (Opnum 11)

The **CloseHost** method closes a specified target.

```
HRESULT CloseHost(
        [in] BSTR HostName);
```

HostName: The name of the target.

Return Values:

The following value is possible:

Return value/code	**Description**
S_OK 0x00000000	The operation completed successfully.

The server SHOULD process this method as follows:

- Finds the specified target in the collection maintained in the **HostList** data object (section 3.1.1.7) and removes it.
- Disconnects all sessions that are connected to the target.
- Removes the assignments of all disk objects that are assigned to the target.
- Unregisters the target with the iSNS manager.

The server SHOULD NOT commit to persist these changes, and only the collection of iSCSI targets in memory SHOULD be affected.

3.27.5 Timer Events

None.

3.27.6 Other Local Events

None.

3.28 IHostMgr2 Client Details

The **IHostMgr2** interface (section 3.27) extends the **IHostMgr** interface (section 3.25) with support for non-persistent iSCSI targets.

3.28.1 Abstract Data Model

The client data model is specified in section 3.2.1.

3.28.2 Timers

None.

3.28.3 Initialization

Client initialization is specified in section 3.2.3.

3.28.4 Message Processing Events and Sequencing Rules

This section specifies client processing of error returns by methods of the **IHostMgr2** interface (section 3.27).

3.28.4.1 NewHost2 (Opnum 8)

Processing of the following error returns SHOULD be performed by client callers of the **NewHost2** method (section 3.27.4.1):

ERROR_FILE_EXISTS

A return value of ERROR_FILE_EXISTS indicates that an attempt has been made to create a new target with the name of a target that already exists.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the specified target name cannot be used.<56>

ERROR_INVALID_DATA

A return value of ERROR_INVALID_DATA indicates that the name specified for the new target is not valid. A possible cause is that the length of the name exceeds an implementation-defined limit.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the specified target name cannot be used.<57>

3.28.5 Timer Events

None.

3.28.6 Other Local Events

None.

3.29 IHostMgr3 Server Details

The **IHostMgr3** interface extends the **IHostMgr2** interface (section 3.27) with support for HA clusters.

3.29.1 Abstract Data Model

The **IHostMgr3** interface provides the means to enumerate iSCSI targets, regardless of their cluster resource state.

For more details concerning the abstract data organization for this interface, see the target management data model (section 3.1.1.7).

3.29.2 Timers

None.

3.29.3 Initialization

Target manager initialization is specified in section 3.1.3.

3.29.4 Message Processing Events and Sequencing Rules

The **IHostMgr3** interface includes the following method:

Method	Description
EnumAllClusterHosts	Returns an enumeration of all targets in an HA cluster environment. Opnum: 12

3.29.4.1 EnumAllClusterHosts (Opnum 12)

The **EnumAllClusterHosts** method returns an enumeration of all targets in an HA cluster environment.

```
HRESULT EnumAllClusterHosts(
        [out] IEnumHost ** EnumHost);
```

EnumHost: A pointer to an **IEnumHost** interface (section 3.8) pointer that returns the enumeration of targets. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumAllClusterHosts** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates or updates a collection of **IHost** interface objects (section 3.21) that define all iSCSI targets that exist on the iSCSI Software Target, and saves the result in a **HostList** data object (section 3.1.1.7).
- Creates an **IEnumHost** interface object and initializes it with the collection of targets.
- Returns a pointer to the **IEnumHost** interface by using the *EnumHost* parameter.

If the ISTM server is not running in an HA cluster environment, only targets with a **ResourceState** data element (section 3.1.1.7) value of **ClusterResourceOnline** ([MS-CMRP] section 3.1.4.1.13) SHOULD be returned by this method. In an HA cluster environment, targets in the **ClusterResourceOffline** state SHOULD also be returned.

3.29.5 Timer Events

None.

3.29.6 Other Local Events

None.

3.30 ILocalDeviceMgr Server Details

The **ILocalDeviceMgr** interface is used to manage devices that are local to the iSCSI Software Target computer, including their configuration information.

3.30.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the local device management data model (section 3.1.1.3).

3.30.2 Timers

None.

3.30.3 Initialization

Local device manager initialization is specified in section 3.1.3.

3.30.4 Message Processing Events and Sequencing Rules

The **ILocalDeviceMgr** interface includes the following methods:

Method	Description
EnumDisks	Returns an enumeration of the disks in the iSCSI Software Target computer. Opnum: 3
EnumVolumes	Returns an enumeration of the volumes that have been configured on the iSCSI Software Target computer. Opnum: 4
CreateVolume	Creates a new volume on the iSCSI Software Target computer. Opnum: 5
IsValidWtDevice	Determines whether a specified file is a VHD file and can be used by the iSCSI Software Target. Opnum: 6
IsDiskPartAvailable	Determines whether the iSCSI Software Target computer can partition disks. Opnum: 7
CreateWtFile	Creates a VHD file that represents a virtual disk. Opnum: 8
GrowWtFile	Increases the size of a specified VHD file. Opnum: 9
DeleteWtFile	Deletes a specified VHD file. Opnum: 10
IsPathValid	Verifies that a specified path is valid. Opnum: 11
IsPathNtfs	Checks a specified path to determine whether it resides on an NTFS volume. Opnum: 12
DoesFileExist	Determines whether a specified file exists in the file system. Opnum: 13
GetDriveInfo	Retrieves information about a specified drive. Opnum: 14
EnumDirectory	Returns an interface for enumerating directories and files that exist in a volume. Opnum: 15
EnumPortals	Returns an enumeration of the portals that exist on the iSCSI Software Target. Opnum: 16
GetPortal	Retrieves information about a specified portal.

Method	Description
	Opnum: 17
SetPortalConfig	Sets configuration values for a specified portal. Opnum: 18
DeviceManagerRescan	Refreshes the device information of the iSCSI Software Target. Opnum: 19
GetProcessors	Retrieves information about the number of processing units on the iSCSI Software Target computer. Opnum: 20

3.30.4.1 EnumDisks (Opnum 3)

The **EnumDisks** method returns an enumeration of the disks in the iSCSI Software Target computer.

```
HRESULT EnumDisks(
        [out] IEnumDisk ** ppEnumDisk);
```

ppEnumDisk: A pointer to an **IEnumDisk** interface (section 3.6) pointer that returns the disk enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumDisks** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates a collection of information for all disks on the iSCSI Software Target computer, with each element of the collection being an instance of a **DISK** structure (section 2.2.4.4).
- Populates the collection using information from the underlying operating system.
- Creates an **IEnumDisk** interface object and MAY associate with it the collection of disk information.<58>

- Returns the **IEnumDisk** interface pointer by using the *ppEnumDisk* parameter.

3.30.4.2 EnumVolumes (Opnum 4)

The **EnumVolumes** method returns an enumeration of the volumes that have been configured on the iSCSI Software Target computer.

```
HRESULT EnumVolumes(
        [out] IEnumVolume ** ppEnumVolume);
```

ppEnumVolume: A pointer to an **IEnumVolume** interface (section 3.17) pointer that returns the volume enumeration. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- For each volume on the iSCSI Software Target computer, the server SHOULD check the following:
- The volume exists on a local fixed drive.
- The volume is not nested within another volume.
- In an HA Cluster configuration, the volume is based on a disk that is owned by the current node.
- Creates a collection of information for all volumes on the iSCSI Software Target computer that meet these criteria, with the elements of the collection defined according to the abstract data model (section 3.1.1.3.2) that is relevant to the **VOLUME** structure (section 2.2.4.19).
- Creates an **IEnumVolume** interface object and associates with it the collection of volume information.
- Returns the **IEnumVolume** interface pointer by using the *ppEnumVolume* parameter.

3.30.4.3 CreateVolume (Opnum 5)

The **CreateVolume** method MAY create a new volume on the iSCSI Software Target computer.<59>

```
HRESULT CreateVolume(
        [in] unsigned long nDiskIndex,
        [in] hyper hyOffset,
        [in] hyper hyLength,
        [in] unsigned long nVolumeType,
        [out] BSTR * DevicePath);
```

nDiskIndex: The operating system index for the disk that hosts the volume.

hyOffset: The offset of the volume from the beginning of the hosting disk, in bytes.

hyLength: The total size of the volume, in bytes.

nVolumeType: The type of volume to create.

DevicePath: A pointer to a string that returns the path of the volume.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.30.4.4 IsValidWtDevice (Opnum 6)

The **IsValidWtDevice** method determines whether a specified file is a VHD file and can be used by the iSCSI Software Target.

```
HRESULT IsValidWtDevice(
        [in] BSTR DevicePath);
```

DevicePath: A string that contains the path to the file in the domain of the file system.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	The supplied file path is malformed or invalid.
ERROR_UNSUPPORTED_TYPE 0x8007065e	The device path is for a VHD type that is not supported.
ERROR_PATH_NOT_FOUND 0x80070003	The system cannot find the path specified.
ERROR_FILE_NOT_FOUND 0x80070002	The system cannot find the file specified.

Return value/code	Description
ERROR_INVALID_DATA 0x8007000d	The file that is specified in the *DevicePath* parameter is not a VHD file or has invalid identifying data, or it is not one of the supported VHD file types.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Checks that the text in the *DevicePath* parameter is a correctly formed path.
- In an HA cluster environment, verifies that the given path exists on one of the cluster's shared disks and that the shared disk is owned by the computer.
- Determines whether the operating system object that is identified in the *DevicePath* parameter is an actual VHD file and that it is one of the supported VHD file types, according to the **VhdType** enumeration (section 2.2.3.9).

3.30.4.5 IsDiskPartAvailable (Opnum 7)

The **IsDiskPartAvailable** method MAY determine whether the iSCSI Software Target computer can partition disks.<60>

```
HRESULT IsDiskPartAvailable(
        [out] boolean * bAvail);
```

bAvail: A pointer to a Boolean value that returns the result of this operation. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_POINTER 0x80004003	Invalid pointer.

When the **IsDiskPartAvailable** method is called, the ISTM server SHOULD check whether the *bAvail* parameter is NULL, and if it is, the server SHOULD return E_POINTER. Otherwise, the server SHOULD make an implementation-specific determination of whether the iSCSI Software Target computer can partition disks and SHOULD return the result by using the *bAvail* parameter.

3.30.4.6 CreateWtFile (Opnum 8)

The **CreateWtFile** method creates a VHD file that represents a virtual disk.

```
HRESULT CreateWtFile(
        [in] BSTR FilePath,
        [in] hyper hyFileSize,
        [in] boolean bSparse);
```

FilePath: A string that contains the path of the file in the domain of the file system.

hyFileSize: The required size of the file, in bytes.

bSparse: A Boolean value that specifies whether the file is to be a sparse file.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	The supplied file path is malformed or invalid.
ERROR_BAD_LENGTH 0x80070018	The specified size of the file is incompatible with implementation limits.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Verifies that the text in the *FilePath* parameter is a correctly formed path. If the text in the *FilePath* parameter identifies an existing operating system object, the server verifies that the path does not identify a directory. If any of these tests fail, the server SHOULD return E_INVALIDARG.
- Checks that the value of the *hyFileSize* parameter is aligned to implementation-specific values. Size alignment to certain block-sizes might allow an implementation to optimize I/O operations. If the size is not compatible with implementation-specific alignment requirements, the server SHOULD return E_INVALIDARG.<61>
- Checks the value of the *hyFileSize* parameter against implementation-specific limits. If the value is less than or greater than those limits, the server SHOULD return ERROR_BAD_LENGTH.<62>
- Creates a VHD file of type **FixedVhd** (section 2.2.3.9) and stores information about that file according to the abstract data model in section 3.1.1.3.3.

Note If a file with the path and name specified in the *FilePath* parameter already exists, the server SHOULD overwrite that file.

3.30.4.7 GrowWtFile (Opnum 9)

The **GrowWtFile** method increases the size of a specified VHD file.

```
HRESULT GrowWtFile(
          [in] BSTR FilePath,
          [in] hyper hyAdditionalSize);
```

FilePath: The path of the file in the domain of the file system.

hyAdditionalSize: The number of bytes that are to be added to the size of the file. The value of this parameter SHOULD be interpreted as an unsigned value.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	The supplied file path is malformed or invalid.
ERROR_BAD_LENGTH 0x80070018	The size of the file specified is incompatible with implementation limits.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Verifies that the text in the *FilePath* parameter is a correctly formed path. The server also verifies that the identified operating system object is an actual VHD file and that it is one of the supported VHD file types according to the **VhdType** enumeration (section 2.2.3.9). As described in the section "Implementing a differencing Hard Disk" of the VHD specification [MSFT-VHD], a resize operation is not meaningful for a differencing VHD, and SHOULD be rejected. If any of these tests fail, the server SHOULD return E_INVALIDARG.
- Checks the value in the *hyAdditionalSize* parameter against implementation-specific block-sizes. If the validation fails, the server SHOULD return E_INVALIDARG.<63>
- Verifies that enough free space is available and that the new size of the file is supported by the operating system.
- Increases the size of the file.

3.30.4.8 DeleteWtFile (Opnum 10)

The **DeleteWtFile** method deletes a specified VHD file.

```
HRESULT DeleteWtFile(
        [in] BSTR FilePath);
```

FilePath: The path of the file in the domain of the file system.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	The supplied file path is malformed or invalid.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Verifies that the text in the *FilePath* parameter is a correctly formed path. If this test fails, the server SHOULD return E_INVALIDARG.
- Deletes the file that is identified by the *FilePath* parameter.

3.30.4.9 IsPathValid (Opnum 11)

The **IsPathValid** method verifies that a specified path is valid.

```
HRESULT IsPathValid(
        [in] BSTR FilePath);
```

FilePath: The path of the file in the domain of the file system.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_FAIL 0x80004005	The supplied file path is malformed or invalid or incompatible with the current HA cluster configuration.
ERROR_DEVICE_IN_USE 0x80070964	The path points to an file already used to expose a LUN.
ERROR_INVALID_DRIVE 0x8007000f	The volume where the file path resides is not valid for the requested operation.
ERROR_PATH_NOT_FOUND 0x80070003	The device path is not valid for the current HA cluster configuration.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Verifies that the text in the *FilePath* parameter is a correctly formed path. If the text in the *FilePath* parameter identifies an existing operating system object, the server verifies that the path does not identify a directory or other system device such as a printer port. If the server is in an HA cluster environment, it verifies that the given path exists on one of the cluster's shared disks and that the disk is currently owned by the computer. If any of these tests fail, the server SHOULD return E_FAIL.

- Verifies that VHD file specified by the *FilePath* parameter is not already being used as a virtual disk; otherwise, the server SHOULD return ERROR_DEVICE_IN_USE.
- Verifies that the given path does not reside on a locally mounted virtual disk or on a nested volume; otherwise the server SHOULD return ERROR_INVALID_DRIVE.
- Verifies that the path resides on a local hard drive or similar fixed storage device, and that the device is ready to be used; otherwise, the server SHOULD return an appropriate error.

3.30.4.10 IsPathNtfs (Opnum 12)

The **IsPathNtfs** method checks a specified path to determine whether it resides on an NTFS volume.

```
HRESULT IsPathNtfs(
        [in] BSTR Path);
```

Path: The path to check.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
S_FALSE 0x00000001	The file path is valid but on an unknown file system or not on an NTFS.
E_INVALIDARG 0x80070057	The supplied file path is malformed or invalid.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Verifies that the text in the *Path* parameter is a correctly formed path; otherwise, the server SHOULD return E_INVALIDARG.
- In an HA cluster environment, verifies that the given path exists on one of the cluster's shared disks; otherwise, the server SHOULD return S_FALSE.
- Finds the volume on which the path is mounted, and obtains the file system information of the volume. If either of these operations fails, the server SHOULD return S_FALSE.
- Returns S_OK if the file system of the volume is NTFS; otherwise the server SHOULD return S_FALSE.

3.30.4.11 DoesFileExist (Opnum 13)

The **DoesFileExist** method determines whether a specified file exists in the file system.

```
HRESULT DoesFileExist(
        [in] BSTR FilePath);
```

FilePath: The full path to the file of interest in the domain of the file system.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	The supplied file path is malformed.
E_FAIL 0x80004005	The file specified by the supplied file path does not exist or the file path is not valid.

The server SHOULD process this method as follows:

- Verifies that the text in the *FilePath* parameter is a correctly formed path; otherwise, the server SHOULD return E_INVALIDARG.
- Queries the file system. If the file exists, the server SHOULD return S_OK; otherwise the server SHOULD return E_FAIL.

3.30.4.12 GetDriveInfo (Opnum 14)

The **GetDriveInfo** method retrieves information about a specified drive.

Note This method is deprecated. The **GetVolumeInfo** method (section 3.32.4.3) SHOULD be used instead.

```
HRESULT GetDriveInfo(
        [in] BSTR Drive,
        [in, out] DRIVE_INFO * pDriveInfo);
```

Drive: The operating system designation of the drive of interest.

pDriveInfo: A pointer to a **DRIVE_INFO** structure (section 2.2.4.6) that returns information about the drive.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Collects information about the specified drive, according to the volume data model (section 3.1.1.3.2), and sets the values in the specified **DRIVE_INFO** structure.
- Returns the drive information in the **DRIVE_INFO** structure by using the *pDriveInfo* parameter.

3.30.4.13 EnumDirectory (Opnum 15)

The **EnumDirectory** method returns an interface for enumerating directories and files that exist in a volume.

```
HRESULT EnumDirectory(
        [out] IDirectoryEnum ** ppDirectoryEnum);
```

ppDirectoryEnum: A pointer to an **IDirectoryEnum** interface (section 3.3) pointer that returns the directory enumeration interface. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The ISTM server creates an instance of the **IDirectoryEnum** interface and returns a pointer to it by using the *ppDirectoryEnum* parameter.

3.30.4.14 EnumPortals (Opnum 16)

The **EnumPortals** method returns an enumeration of the portals [RFC3720] that exist on the iSCSI Software Target.

```
HRESULT EnumPortals(
        [out] IEnumPortal ** ppEnumPortal);
```

ppEnumPortal: A pointer to an **IEnumPortal** interface (section 3.11) pointer that returns the portal enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumPortals** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates a collection of information for all portals used by the iSCSI Software Target, with the elements of the collection defined according to the abstract data model (section 3.1.1.3.5) that is relevant to the **PORTAL** structure (section 2.2.4.13).
- Creates an **IEnumPortal** interface object and associates with it the collection of portal information.
- Returns the **IEnumPortal** interface pointer by using the *ppEnumPortal* parameter.

In an HA Cluster configuration, all of the portals that belong to the computer are returned, regardless of resource group membership.

3.30.4.15 GetPortal (Opnum 17)

The **GetPortal** method retrieves information about a specified portal.

```
HRESULT GetPortal(
        [in] BSTR IpAddress,
        [out] PORTAL * pPortal);
```

IpAddress: The string representation of the IP address of the network adapter that the portal of interest is associated with. For a particular portal, this SHOULD be a value that was obtained during portal enumeration using the **IEnumPortal** interface (section 3.11).

pPortal: A pointer to a **PORTAL** structure (section 2.2.4.132) that returns information about the portal. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	
ERROR_FILE_NOT_FOUND 0x80070002	The supplied portal identifier is not found.

The server SHOULD process this method as follows:

- Uses the value of the *IpAddress* parameter to find information about the portal, according to the abstract data model (section 3.1.1.3.5) that is relevant to the **PORTAL** structure.
- Returns the information to the caller by using the *pPortal* parameter.

3.30.4.16 SetPortalConfig (Opnum 18)

The **SetPortalConfig** method sets configuration values for a specified portal.

```
HRESULT SetPortalConfig(
        [in] BSTR IpAddress,
        [in] unsigned long nPort,
        [in] boolean bUse);
```

IpAddress: The string representation of the IP address of the network adapter that the portal of interest is associated with. For a particular portal, this SHOULD be a value that was obtained during portal enumeration using the **IEnumPortal** interface (section 3.11).

nPort: The TCP port number that is to be used for iSCSI messages coming through this portal. The value MUST be greater than zero and less than or equal to 65535.

bUse: A Boolean value that determines whether the iSCSI Software Target will listen for iSCSI messages on this IP address/port pair.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	The supplied IP protocol port number is invalid.
ERROR_FILE_NOT_FOUND 0x80070002	The supplied portal identifier is not found.

The server SHOULD process this method as follows:

- Uses the value of the *IpAddress* parameter to find the portal of interest.
- Checks the *nPort* parameter to determine if the value is within the acceptable range, and sets the portal's TCP port number to that value.

- Sets the portal's **ListenFlag** data element (section 3.1.1.3.5) to the value of the *bUse* parameter.
- If the **ListenFlag** data element changes as a result of this operation, the server SHOULD notify the iSNS manager of the change in the status of the available portals.

3.30.4.17 DeviceManagerRescan (Opnum 19)

The **DeviceManagerRescan** method refreshes the device information of the iSCSI Software Target so that the operating system is aware of the storage devices that have been mounted.

Note This method can be called by an ISTM client as part of other management operations that involve changing or modifying the local mount status (**MountType**, section 2.2.3.6) of LUNs when a virtual disk or snapshot is mounted.

```
HRESULT DeviceManagerRescan(
        [in] boolean bSynchronous);
```

bSynchronous: A Boolean value that indicates whether the refresh operation is to be synchronous. If this value is true, the server SHOULD NOT send a response until the refresh operation is complete; otherwise, the server SHOULD send a response immediately.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

When the **DeviceManagerRescan** method is called, the ISTM server instructs the operating system to initiate a refresh of the device information. The server SHOULD returns S_OK either immediately or after the refresh operation is complete, depending on the value of the *bSynchronous* parameter.

3.30.4.18 GetProcessors (Opnum 20)

The **GetProcessors** method MAY retrieve information about the number of processing units on the iSCSI Software Target computer.<64>

```
HRESULT GetProcessors(
        [out] unsigned long * pProcessors);
```

pProcessors: A pointer to an unsigned value that returns information about the number of processors on the iSCSI Software Target computer that the iSCSI Software Target process can utilize. This value is expressed as bit settings in a 32-bit bitmask.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.30.5 Timer Events

None.

3.30.6 Other Local Events

None.

3.31 ILocalDeviceMgr Client Details

The **ILocalDeviceMgr** interface (section 3.30) is used to manage devices that are local to the iSCSI Software Target computer, including their configuration information.

3.31.1 Abstract Data Model

The client data model is specified in section 3.2.1.

3.31.2 Timers

None.

3.31.3 Initialization

Client initialization is specified in section 3.2.3.

3.31.4 Message Processing Events and Sequencing Rules

This section specifies client processing of error returns that the ISTM client receives from the ISTM server as a result of calling specific methods of the **ILocalDeviceMgr** interface (section 3.30).

3.31.4.1 IsValidWtDevice (Opnum 6)

Processing of the following error responses SHOULD be performed by client callers of the **IsValidWtDevice** method (section 3.30.4.4):

ERROR_PATH_NOT_FOUND

A return value of ERROR_PATH_NOT_FOUND indicates that the *DevicePath* parameter specifies the root directory of a fixed disk.

The ISTM client SHOULD take an implementation-defined action to inform the user that this particular path is not allowed.<65>

ERROR_FILE_NOT_FOUND

A return value of ERROR_FILE_NOT_FOUND indicates that the VHD file that is specified in the *DevicePath* parameter does not exist.

The ISTM client SHOULD take an implementation-defined action to inform the user that this particular file does not exist.<66>

ERROR_INVALID_DATA

A return value of ERROR_INVALID_DATA indicates that the file that is specified in the *DevicePath* parameter is not a VHD file, is an un supported VHD file type, or has invalid data in its VHD footer.

The ISTM client SHOULD take an implementation-defined action to inform the user that the specified file is not appropriate for use as a virtual disk.<67>

3.31.4.2 CreateWtFile (Opnum 8)

Processing of the following error response SHOULD be performed by client callers of the **CreateWtFile** method (section 3.30.4.6):

ERROR_BAD_LENGTH

A return value of ERROR_BAD_LENGTH indicates that the specified size of the file is incompatible with implementation limits.

The ISTM client SHOULD take an implementation-defined action to inform the user that the specified size is not allowed.<68>

3.31.4.3 IsPathValid (Opnum 11)

Processing of the following error responses SHOULD be performed by client callers of the **IsPathValid** method (section 3.30.4.9):

ERROR_DEVICE_IN_USE

A return value of ERROR_DEVICE_IN_USE indicates that the path specified in the *FilePath* parameter points to a file that is already being used to expose a LUN.

The ISTM client SHOULD take an implementation-defined action to inform the user that the path is in use.<69>

ERROR_INVALID_DRIVE

A return value of ERROR_INVALID_DRIVE indicates that the volume where the file path specified in the *FilePath* parameter resides is not valid for the requested operation.

The ISTM client SHOULD take an implementation-defined action to inform the user that the volume cannot be used.<70>

3.31.5 Timer Events

None.

3.31.6 Other Local Events

None.

3.32 ILocalDeviceMgr2 Server Details

The **ILocalDeviceMgr2** interface extends the **ILocalDeviceMgr** interface (section 3.30) with support for HA clusters.

3.32.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the local device management data model (section 3.1.1.3).

3.32.2 Timers

None.

3.32.3 Initialization

Local device manager initialization is specified in section 3.1.3.

3.32.4 Message Processing Events and Sequencing Rules

The **ILocalDeviceMgr2** interface includes the following methods:

Method	Description
EnumDisks2	Returns an enumeration of the disks in the iSCSI Software Target computer. Opnum: 21
EnumVolumes2	Returns an enumeration of the volumes that have been configured on the iSCSI Software Target computer. Opnum: 22
GetVolumeInfo	Retrieves information about a particular volume. Opnum: 23
EnumResourceGroup	Returns an enumeration of all resource groups that relate to the iSCSI Software Target computer. Opnum: 24
EnumLocalResourceGroup	Returns an enumeration of all resource groups that are owned by the iSCSI Software Target computer. Opnum: 25
GetResourceGroupGuid	Retrieves the GUID of a particular resource group. Opnum: 26
GetResourceGroupFromPath	Retrieves the name of a resource group that a specified path is part of. Opnum: 27
RenameResourceGroup	Renames a specified resource group. Opnum: 28
EnumLocalPortals	Returns an enumeration of the portals owned by the iSCSI Software Target computer that are part of a specified resource group. Opnum: 29
GetResourceGroupOwner	Retrieves the name of the computer that owns the resources in a specified resource group. Opnum: 30

3.32.4.1 EnumDisks2 (Opnum 21)

The **EnumDisks2** method returns an enumeration of the disks in the iSCSI Software Target computer.

```
HRESULT EnumDisks2(
        [out] IEnumDisk2 ** ppEnumDisk);
```

ppEnumDisk: A pointer to an **IEnumDisk2** interface (section 3.7) pointer that returns the disk enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible.

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumDisks2** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates a collection of information for all disks on the iSCSI Software Target computer, with each element of the collection being an instance of **DISK2** structure (section 2.2.4.5).
- Populates the collection using information from the underlying operating system.
- Creates an **IEnumDisk2** interface object and MAY associate with it the collection of disk information.<71>
- Returns the **IEnumDisk2** interface pointer by using the *ppEnumDisk* parameter.

3.32.4.2 EnumVolumes2 (Opnum 22)

The **EnumVolumes2** method returns an enumeration of the volumes that have been configured on the iSCSI Software Target computer.

```
HRESULT EnumVolumes2(
        [out] IEnumVolume2 ** ppEnumVolume);
```

ppEnumVolume: A pointer to an **IEnumVolume2** interface (section 3.18) pointer that returns the volume enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumVolumes2** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- For each volume on the iSCSI Software Target computer, the server SHOULD check the following:
- The volume exists on a local fixed drive.
- The volume is not nested within another volume.
- In an HA cluster configuration, the volume belongs to the resource group of the HA cluster.
- Creates a collection of information for all volumes on the iSCSI Software Target computer that meet these criteria, with the elements of the collection defined according to the abstract data model (section 3.1.1.3.2) that is relevant to the **VOLUME2** structure (section 2.2.4.20).
- Creates an **IEnumVolume2** interface object and associates with it the collection of volume information.
- Returns the **IEnumVolume2** interface pointer by using the *ppEnumVolume* parameter.

3.32.4.3 GetVolumeInfo (Opnum 23)

The **GetVolumeInfo** method retrieves information about a particular volume.

```
HRESULT GetVolumeInfo(
        [in] BSTR VolumePath,
        [in, out] VOLUME2 * pVolumeInfo);
```

VolumePath: An operating system path that resides on the volume. This is typically the value of the **szMountPoint** member of the **VOLUME2** structure (section 2.2.4.20) that was returned as part of a volume enumeration.

pVolumeInfo: A pointer to the **VOLUME2** structure that returns information about the volume.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_FILE_NOT_FOUND 0x80070002	The specified volume is not found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Finds the mount point of the volume on which the path specified in the *VolumePath* parameter resides.
- Searches for the volume in an enumeration of volumes that is similar to that created by **EnumVolumes2** (section 3.32.4.2), with the same criteria that are used when processing that method.
- If the specified volume is found, returns the volume information according to the abstract data model (section 3.1.1.3.2) that is relevant to the **VOLUME2** structure (section 2.2.4.20).
- If the specified volume is not found, the server SHOULD return ERROR_FILE_NOT_FOUND.

3.32.4.4 EnumResourceGroup (Opnum 24)

The **EnumResourceGroup** method returns an enumeration of all resource groups that relate to the iSCSI Software Target computer.

```
HRESULT EnumResourceGroup(
       [out] IEnumResourceGroup ** ppEnumResourceGroup);
```

ppEnumResourceGroup: A pointer to an **IEnumResourceGroup** interface (section 3.12) pointer that returns the resource group enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED	General access denied error.

Return value/code	Description
0x80070005	
E_NOTIMPL 0x80004001	The machine is not configured in an HA cluster.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumResourceGroup** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates a collection of information for all resource groups that relate to the HA cluster configuration of the iSCSI Software Target computer, with the elements of the collection defined according to the abstract data model (section 3.1.1.3.7) that is relevant to the **RESOURCE_GROUP** structure (section 2.2.4.14).
- Creates an **IEnumResourceGroup** interface object and associates the resource group information for all of the resource groups with it.
- Returns the **IEnumResourceGroup** interface pointer by using the *ppEnumResourceGroup* parameter.

3.32.4.5 EnumLocalResourceGroup (Opnum 25)

The **EnumResourceGroup** method returns an enumeration of the resource groups that are owned by the iSCSI Software Target computer.

```
HRESULT EnumLocalResourceGroup(
        [out] IEnumResourceGroup ** ppEnumResourceGroup);
```

ppEnumResourceGroup: A pointer to an **IEnumResourceGroup** interface (section 3.12) pointer that returns the resource group enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_NOTIMPL 0x80004001	The machine is not configured in an HA cluster.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumLocalResourceGroup** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Enumerates all resource groups that relate to the HA cluster configuration of the iSCSI Software Target computer, similar to the processing for the **IEnumResourceGroup** method (section 3.32.4.4).
- Creates a collection of information for all resource groups that are owned by the iSCSI Software Target computer, with the elements of the collection defined according to the abstract data model (section 3.1.1.3.7) that is relevant to the **RESOURCE_GROUP** structure (section 2.2.4.14).
- Creates an **IEnumResourceGroup** interface object and associates with it the resource group information for all of the resulting resource groups.
- Returns the **IEnumResourceGroup** interface pointer by using the *ppEnumResourceGroup* parameter.

3.32.4.6 GetResourceGroupGuid (Opnum 26)

The **GetResourceGroupGuid** method retrieves the GUID of a particular resource group.

```
HRESULT GetResourceGroupGuid(
        [in] BSTR ResourceGroup,
        [out] GUID * pGuid);
```

ResourceGroup: The name of the resource group of interest. The value corresponds to the **ResourceGroupName** data element (section 3.1.1.3.7) of the resource group. The string MUST NOT be empty.

pGuid: A pointer to a **GUID** structure (section 2.2) that returns the GUID of the resource group. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	One or more arguments are invalid.
E_POINTER 0x80004003	Invalid pointer.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Verifies that the *ResourceGroup* parameter is not empty and that the *pGuid* parameter is not NULL; otherwise the server SHOULD return E_INVALIDARG or E_POINTER, respectively.
- Obtains the GUID of the resource group identified by the *ResourceGroup* parameter.
- Returns the GUID by using the *pGuid* parameter.

3.32.4.7 GetResourceGroupFromPath (Opnum 27)

The **GetResourceGroupFromPath** method retrieves the name of the resource group that a specified path is part of.

```
HRESULT GetResourceGroupFromPath(
        [in] BSTR FilePath,
        [out] BSTR * pResourceGroup);
```

FilePath: The path on which to base the search for the resource group. The value can represent the path to a directory or to a file.

pResourceGroup: A pointer to a string that returns the name of the resource group. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	The supplied path is malformed or invalid.
E_POINTER 0x80004003	Invalid pointer.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Verifies that the text in the *FilePath* parameter is a correctly formed path; otherwise, the server SHOULD return E_INVALIDARG.
- Verifies that the *pResourceGroup* parameter is not NULL; otherwise, the server SHOULD return E_POINTER.
- Determines which local disk contains the path specified by the *FilePath* parameter.
- Determines which resource group owns the local disk.
- Returns the name of the resource group, as defined by the **ResourceGroupName** data element (section 3.1.1.3.7), by using the *pResourceGroup* parameter.

3.32.4.8 RenameResourceGroup (Opnum 28)

The **RenameResourceGroup** method renames a specified resource group.

```
HRESULT RenameResourceGroup(
        [in] BSTR ResourceGroup,
        [in] BSTR NewResourceGroup);
```

ResourceGroup: The name of the resource group to rename. This value corresponds to the **ResourceGroupName** data element (section 3.1.1.3.7) of the resource group. This string MUST NOT be empty.

NewResourceGroup: The name that the resource group will be given as a result of this operation. The string MUST NOT be empty.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	One or both of the supplied names is invalid.
ERROR_INVALID_HANDLE 0x80070006	The machine is not configured in an HA cluster.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Validates the strings in the *ResourceGroup* and *NewResourceGroup* parameters; if either is invalid, the server SHOULD return E_INVALIDARG.
- Finds the resource group that is identified by the *ResourceGroup* parameter.
- Changes the name of the resource group to the value specified in the *NewResourceGroup* parameter.

3.32.4.9 EnumLocalPortals (Opnum 29)

The **EnumLocalPortals** method returns an enumeration of the portals owned by the iSCSI Software Target computer that are part of a specified resource group.

```
HRESULT EnumLocalPortals(
        [in] BSTR ResourceGroup,
        [out] IEnumPortal ** ppEnumPortal);
```

ResourceGroup: The name of the resource group that contains the portals to enumerate. This value corresponds to the **ResourceGroupName** data element (section 3.1.1.3.7) of the resource group.

ppEnumPortal: A pointer to an **IEnumPortal** interface (section 3.11) pointer that returns the portal enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_POINTER 0x80004003	Invalid pointer.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumLocalPortals** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Verifies that the *ppEnumPortal* parameter is not NULL; otherwise, the server SHOULD return E_POINTER.
- If the iSCSI Software Target computer is not part of an HA cluster, the server SHOULD processes this method in a manner similar to the **EnumPortals** method (section 3.30.4.14).
- Otherwise, the server finds all of the portals that meet the following criteria:
- The portal exists in the same HA cluster environment as the iSCSI Software Target computer.
- If the *ResourceGroup* parameter is not empty, the portal is part of the specified resource group.
- The portal resource is owned by the iSCSI Software Target computer.
- The portal is online, as indicated by the **ClusterResourceOnline** value ([MS-CMRP] section 3.1.4.1.13) of the cluster resource state.

Note If the portal is in the process of coming online, as indicated by the **ClusterResourceOnlinePending** value of the cluster resource state, the server MUST wait until the portal is either online or has failed to come online.

- Creates a collection of information for portals that meet these criteria, with the elements of the collection defined according to the abstract data model (section 3.1.1.3.5) that is relevant to the **PORTAL** structure (section 2.2.4.13).
- Creates an **IEnumPortal** interface object and associates with it the collection portal information.
- Returns the **IEnumPortal** interface pointer by using the *ppEnumPortal* parameter.

Note During the processing of this method, the server MUST enumerate the portals associated with both the IPv4 and IPv6 forms of IP addresses.

3.32.4.10 GetResourceGroupOwner (Opnum 30)

The **GetResourceGroupOwner** method retrieves the name of the computer that owns the resources in a specified resource group.

```
HRESULT GetResourceGroupOwner(
        [in] BSTR ResourceGroup,
        [out] BSTR * pOwner);
```

ResourceGroup: The name of the resource group that contains the resources. This value corresponds to the **ResourceGroupName** data element (section 3.1.1.3.7) of the resource group. The string MUST NOT be empty.

pOwner: A pointer to a string that returns the name of the owning computer. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_POINTER 0x80004003	One or both of the supplied names or pointers is invalid.
ERROR_INVALID_HANDLE 0x80070006	The machine is not configured in an HA cluster.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Verifies the parameters, and if either is invalid, the server SHOULD return E_POINTER.
- Finds the name of the computer node that owns the resources in the resource group specified by the *ResourceGroup* parameter.
- Returns the name of the computer by using the *pOwner* parameter.

3.32.5 Timer Events

None.

3.32.6 Other Local Events

None.

3.33 ILocalDeviceMgr3 Server Details

The **ILocalDeviceMgr3** interface extends the **ILocalDeviceMgr2** interface (section 3.32) with support for differencing VHD files.

3.33.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the local device management data model (section 3.1.1.3).

3.33.2 Timers

None.

3.33.3 Initialization

Local device manager initialization is specified in section 3.1.3.

3.33.4 Message Processing Events and Sequencing Rules

The **ILocalDeviceMgr3** interface includes the following method:

Method	Description
CreateDiffWtFile	Creates a differencing VHD file that is based on a specified parent VHD file. Opnum: 31

3.33.4.1 CreateDiffWtFile (Opnum 31)

The **CreateDiffWtFile** method creates a differencing VHD file that is based on a specified parent VHD file.

```
HRESULT CreateDiffWtFile(
        [in] BSTR FilePath,
        [in] BSTR ParentPath);
```

FilePath: The path of the file to be created, in the domain of the file system. This parameter MUST NOT be NULL.

ParentPath: The path of the parent VHD file, in the domain of the file system. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	The supplied file path is malformed or invalid.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Verifies that the input parameters are not NULL and that the text in the input parameters represents correctly formed paths. If the text in the *FilePath* parameter identifies an existing operating system object, the server verifies that the path does not identify a directory. If any of these tests fail, the server returns E_INVALIDARG.
- Verifies the following:
- The parent VHD file is of type **FixedVhd** (section 2.2.3.9).
- The total size of the parent VHD file plus the size of the metadata of the differencing VHD file is within the file-size limitations of the operating system.<72>
- Creates a VHD file of type **DifferencingVhd** (section 2.2.3.9), as specified in [MSFT-VHD].

Note There are no data blocks written to the differencing VHD file. The data blocks are written as needed when data is written to the virtual disk; that is, the combination of the parent VHD file and the differencing VHD file.

Note If a file with the path and name specified in the *FilePath* parameter already exists, the server SHOULD overwrite that file.

3.33.5 Timer Events

None.

3.33.6 Other Local Events

None.

3.34 ISessionManager Server Details

The **ISessionManager** interface is used to obtain information about iSCSI sessions.

3.34.1 Abstract Data Model

The **ISessionManager** interface provides a mechanism to enumerate iSCSI sessions at the global level for the iSCSI Software Target. Only basic iSCSI-protocol-related information is returned.

For more details concerning the abstract data organization for this interface, see the session management data model for session data (section 3.1.1.2.1).

3.34.2 Timers

None.

3.34.3 Initialization

Session manager initialization is specified in section 3.1.3.

3.34.4 Message Processing Events and Sequencing Rules

The **ISessionManager** interface includes the following method:

Method	Description
EnumSessions	Returns an enumeration of all valid iSCSI sessions on the iSCSI Software Target. Opnum: 3

3.34.4.1 EnumSessions (Opnum 3)

The **EnumSessions** method returns an enumeration of all valid iSCSI sessions on the iSCSI Software Target.

```
HRESULT EnumSessions(
        [out] IEnumSession ** ppEnumSession);
```

ppEnumSession: A pointer to an **IEnumSession** interface (section 3.13) pointer that returns the session enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_POINTER 0x80004003	Invalid pointer.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumSessions** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates a collection of information for all valid sessions, with the elements of the collection defined according to the abstract data model (section 3.1.1.2.1) that is relevant to the **SESSION** structure (section 2.2.4.15). A valid iSCSI session is one that has advanced far enough in the protocol login negotiation such that the initiator has provided the IQN.
- Creates an **IEnumSession** interface object and associates with it the session information for all valid sessions.
- Returns the **IEnumSession** interface pointer by using the *ppEnumSession* parameter.

3.34.5 Timer Events

None.

3.34.6 Other Local Events

None.

3.35 ISessionManager2 Server Details

The **ISessionManager2** interface is used to obtain information about iSCSI sessions and iSCSI connections.

3.35.1 Abstract Data Model

The **ISessionManager2** interface provides a mechanism to locate and enumerate iSCSI sessions and iSCSI connections at the global level for the iSCSI Software Target.

For more details concerning the abstract data organization for this interface, see the session management data model (section 3.1.1.2).

3.35.2 Timers

None.

3.35.3 Initialization

Session manager initialization is specified in section 3.1.3.

3.35.4 Message Processing Events and Sequencing Rules

The **ISessionManager2** interface includes the following methods:

Method	Description
EnumSessions2	Returns an enumeration of all valid iSCSI sessions on the iSCSI Software Target. Opnum: 4
EnumConnections	Returns an enumeration of the iSCSI connections that are associated with a specified iSCSI session. Opnum: 5
GetSession	Returns information about a particular iSCSI session. Opnum: 6

Method	Description
GetConnection	Returns information about a particular iSCSI connection that is associated with a specified iSCSI session. Opnum: 7

3.35.4.1 EnumSessions2 (Opnum 4)

The **EnumSessions2** method returns an enumeration of all valid iSCSI sessions on the iSCSI Software Target.

```
HRESULT EnumSessions2(
        [out] IEnumSession2 ** ppEnumSession);
```

ppEnumSession: A pointer to an **IEnumSession2** interface (section 3.14) pointer that returns the session enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_POINTER 0x80004003	Invalid pointer.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumSessions2** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates a collection of information for all valid sessions, with the elements of the collection defined according to the abstract data model (section 3.1.1.2.1) that is relevant to the **SESSION2** structure (section 2.2.4.16). A valid iSCSI session is one that has advanced far enough in the protocol login negotiation such that the initiator has provided the IQN.
- Creates an **IEnumSession2** interface object and associates with it the session information for all valid sessions.
- Returns the **IEnumSession2** interface pointer by using the *ppEnumSession* parameter.

3.35.4.2 EnumConnections (Opnum 5)

The **EnumConnections** method returns an enumeration of the iSCSI connections that are associated with a specified iSCSI session.

```
HRESULT EnumConnections(
        [in] WORD TSIH,
        [out] IEnumConnection ** ppEnumConnection);
```

TSIH: The target session identifying handle (TSIH) [RFC3720]. This value MUST NOT be zero.

ppEnumConnection: A pointer to an **IEnumConnection** interface (section 3.5) pointer that returns the connection enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	The supplied target session identifying handle (TSIH) is invalid.
E_POINTER 0x80004003	Invalid pointer.
ERROR_FILE_NOT_FOUND 0x80070002	The supplied TSIH is not found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumConnections** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- If the *TSIH* parameter is set to zero, the server returns E_INVALIDARG.
- Otherwise, for the iSCSI session specified by the *TSIH* parameter, the server creates a collection of information about the iSCSI connections that are associated with that session, with the elements of the collection defined according to the abstract data model (section 3.1.1.2.2) that is relevant to the **CONNECTION** structure (section 2.2.4.2).
- Creates an **IEnumConnection** interface object and associates the connection information with it.

- Returns the **IEnumConnection** interface pointer by using the *ppEnumConnection* parameter.

3.35.4.3 GetSession (Opnum 6)

The **GetSession** method returns information about a particular iSCSI session.

```
HRESULT GetSession(
        [in] WORD TSIH,
        [out] SESSION2 * pSessionInfo);
```

TSIH: The target session identifying handle (TSIH) [RFC3720] The value MUST NOT be zero.

pSessionInfo: A pointer to a **SESSION2** structure (section 2.2.4.16) that returns the session information. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	The supplied target session identifying handle (TSIH) is invalid.
E_POINTER 0x80004003	Invalid pointer.
ERROR_FILE_NOT_FOUND 0x80070002	The supplied TSIH is not found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **GetSession** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- If the *TSIH* parameter is set to zero, the serer returns E_INVALIDARG.
- Otherwise, the server finds the iSCSI session specified by the *TSIH* parameter.
- Collects information about that session according to the abstract data model in section 3.1.1.2.1 that is relevant to the **SESSION2** structure
- Returns the information by using the *pSessionInfo* parameter.

3.35.4.4 GetConnection (Opnum 7)

The **GetConnection** method returns information about a particular iSCSI connection that is associated with a specified iSCSI session.

```
HRESULT GetConnection(
        [in] WORD TSIH,
        [in] DWORD CID,
        [out] CONNECTION * pConnectionInfo);
```

TSIH: The target session identifying handle (TSIH) [RFC3720] that specifies the iSCSI session. The value MUST NOT be zero.

CID: The connection ID (CID) [RFC3720] of the iSCSI connection.

pConnectionInfo: A pointer to a **CONNECTION** structure (section 2.2.4.2) that returns the connection information. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	The supplied TSIH is invalid.
E_POINTER 0x80004003	Invalid pointer.
ERROR_FILE_NOT_FOUND 0x80070002	The supplied TSIH or connection ID (CID) is not found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **GetConnection** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- If the *TSIH* parameter is set to zero, the serer returns E_INVALIDARG.
- Otherwise, the server finds the iSCSI session specified by the *TSIH* parameter and the iSCSI connection specified by the *CID* parameter.

- Collects information about that connection according to the abstract data model (section 3.1.1.2.2) that is relevant to the **CONNECTION** structure.
- Returns the connection information by using the *pConnectionInfo* parameter.

3.35.5 Timer Events

None.

3.35.6 Other Local Events

None.

3.36 ISnapshot Server Details

The **ISnapshot** interface enables ISTM to manage a particular snapshot.

3.36.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the snapshot management data model (section 3.1.1.5).

3.36.2 Timers

None.

3.36.3 Initialization

Snapshot manager initialization is specified in section 3.1.3.

3.36.4 Message Processing Events and Sequencing Rules

The **ISnapshot** interface includes the following methods:

Method	Description
[propget]Id	Returns the snapshot identifier. Opnum: 3
[propput]Id	Sets the snapshot identifier. Opnum: 4
[propput]VssSnapshotId	Sets the identifier of the snapshot as known to the **volume shadow copy service (VSS)**. Opnum: 5
[propput]VssSnapshotSetId	Sets the identifier of the snapshot set as known to the volume shadow copy service (VSS). Opnum: 6
[propget]OrigWTD	Returns the original disk index of the LUN that corresponds to the snapshot. Opnum: 7
[propput]OrigWTD	Sets the original disk index of the LUN that corresponds to the snapshot.

Method	Description
	Opnum: 8
[propget]TimeStamp	Returns the time stamp of the snapshot. Opnum: 9
[propget]ExportedWTD	Returns the exported disk index of the LUN that corresponds to the snapshot. Opnum: 10
[propput]ExportedWTD	Sets the exported disk index of the LUN that corresponds to the snapshot. Opnum: 11
[propget]Persistent	Returns whether the snapshot is persistent across a system restart. Opnum: 12
[propget]Status	Returns the current status of the snapshot. Opnum: 13
[propput]Status	Sets the current status of the snapshot. Opnum: 14
Save	Saves the state of the snapshot to persistent storage. Opnum: 15
Rollback	Uses the snapshot to roll back the contents of the associated disk. Opnum: 16
AbortRollback	Stops a snapshot rollback that was started with the **Rollback** method. Opnum: 17
ExportAsWTDisk	Used to export the snapshot of a LUN as a VHD disk. Opnum: 18
GetVdsLunInfo	Returns information about the VHD disk LUN for the exported snapshot Opnum: 19
DVMount	Used to surface the snapshot for read-only access on the iSCSI Software Target computer. Opnum: 20
DVDismount	Used to revert the action performed by the **DVMount** method. Opnum: 21

3.36.4.1 [propget]Id (Opnum 3)

The **[propget]Id** method returns the snapshot identifier.

```
[propget] HRESULT Id(
        [out] [retval] GUID * pId);
```

pId: A pointer to a **GUID** structure (section 2.2) that returns the identifier. This value MUST be unique to a particular snapshot.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Id** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD return the GUID from the **Id** element of the **Snapshot** object in the snapshot management abstract data model (section 3.1.1.5) by using the *pId* parameter.

3.36.4.2 [propput]Id (Opnum 4)

The **[propput]Id** method MAY set the snapshot identifier.<73>

```
[propput] HRESULT Id(
        [in] GUID Id);
```

Id: A **GUID** structure (section 2.2) that uniquely identifies the snapshot.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.36.4.3 [propput]VssSnapshotId (Opnum 5)

The **[propput]VssSnapshotId** method MAY set the identifier of the snapshot as known to the volume shadow copy service (VSS).<74>

```
[propput] HRESULT VssSnapshotId(
        [in] GUID Id);
```

Id: A **GUID** structure (section 2.2) that uniquely identifies the snapshot.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.36.4.4 [propput]VssSnapshotSetId (Opnum 6)

The **[propput]VssSnapshotSetId** method MAY set the identifier of the snapshot set as known to the volume shadow copy service (VSS).<75>

```
[propput] HRESULT VssSnapshotSetId(
        [in] GUID Id);
```

Id: A **GUID** structure (section 2.2) that uniquely identifies the snapshot set.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.36.4.5 [propget]OrigWTD (Opnum 7)

The **[propget]OrigWTD** method returns the original disk index of the LUN that corresponds to the snapshot.

```
[propget] HRESULT OrigWTD(
        [out] [retval] unsigned long * pWTD);
```

pWTD: A pointer to a value that returns the original disk index of the LUN.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED	General access denied error.

Return value/code	Description
0x80070005	

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]OrigWTD** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD return the disk index from the **OrigWTD** element of the **Snapshot** object in the snapshot management abstract data model (section 3.1.1.5) by using the *pWTD* parameter.

3.36.4.6 [propput]OrigWTD (Opnum 8)

The **[propput]OrigWTD** method MAY set the original disk index of the LUN that corresponds to the snapshot.<76>

```
[propput] HRESULT OrigWTD(
        [in] unsigned long nWTD);
```

nWTD: The original disk index of the LUN.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.36.4.7 [propget]TimeStamp (Opnum 9)

The **[propget]TimeStamp** method returns the time stamp of the snapshot.

```
[propget] HRESULT TimeStamp(
        [out] [retval] hyper * pTimeStamp);
```

pTimeStamp: A pointer to a value that returns the time stamp in **FILETIME** format (section 2.2).

Return Values:

The following value are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY	The device is not ready.

Return value/code	Description
0x80070015	
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]TimeStamp** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD return the time stamp from the **TimeStamp** element of the **Snapshot** object in the snapshot management abstract data model (section 3.1.1.5) by using the *pTimeStamp* parameter.

3.36.4.8 [propget]ExportedWTD (Opnum 10)

The **[propget]ExportedWTD** method returns the exported disk index of the LUN that corresponds to the snapshot.

```
[propget] HRESULT ExportedWTD(
        [out] [retval] unsigned long * pWTD);
```

pWTD: A pointer to a value that returns the exported disk index of the LUN.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]ExportedWTD** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD return the disk index from the **ExportedWTD** element of the **Snapshot** object in the snapshot management abstract data model (section 3.1.1.5) by using the *pWTD* parameter.

3.36.4.9 [propput]ExportedWTD (Opnum 11)

The **[propput]ExportedWTD** method MAY set the exported disk index of the LUN that corresponds to the snapshot.<77>

```
[propput] HRESULT ExportedWTD(
        [in] unsigned long nWTD);
```

nWTD: The exported disk index of the LUN.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.36.4.10 [propget]Persistent (Opnum 12)

The **[propget]Persistent** method returns whether the snapshot is persistent across a system restart.

```
[propget] HRESULT Persistent(
        [out] [retval] boolean * pPersistent);
```

pPersistent: A pointer to a Boolean value that returns the persistence property of the snapshot.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Persistent** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD return the value of the **Persistent** element of the **Snapshot** object in the snapshot management abstract data model (section 3.1.1.5) by using the *pPersistent* parameter.

3.36.4.11 [propget]Status (Opnum 13)

The **[propget]Status** method returns the current status of the snapshot.

```
[propget] HRESULT Status(
        [out] [retval] unsigned long * pStatus);
```

pStatus: A pointer to a value that returns the current status of the snapshot.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_FILE_NOT_FOUND 0x80070002	The system cannot find the file specified.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Status** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD return the snapshot status from the **Status** element of the **Snapshot** object in the snapshot management abstract data model (section 3.1.1.5) by using the *pStatus* parameter.

3.36.4.12 [propput]Status (Opnum 14)

The **[propput]Status** method MAY set the current status of the snapshot.<78>

```
[propput] HRESULT Status(
        [in] unsigned long nStatus);
```

nStatus: The current status of the snapshot.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.36.4.13 Save (Opnum 15)

The **Save** method saves snapshot properties to persistent storage.

```
HRESULT Save();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **Save** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD save to persistent storage all relevant snapshot elements specified in the snapshot management data model (section 3.1.1.7) and return S_OK.

3.36.4.14 Rollback (Opnum 16)

The **Rollback** method uses the snapshot to roll back the contents of the associated disk.

```
HRESULT Rollback();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY	The device is not ready.

Return value/code	Description
0x80070015	
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_FILE_NOT_FOUND 0x80070002	The system cannot find the file specified.
ERROR_SHARING_VIOLATION 0x80070020	The LUN might be in use by an initiator or locally mounted.
ERROR_INVALID_HANDLE 0x80070006	The LUN have been administratively disabled.
ERROR_INVALID_FUNCTION 0x80070001	Incorrect function.
ERROR_IO_PENDING 0x800703e5	A rollback for the same snapshot might be currently in progress.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Rollback** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Gets access to the device. The rollback SHOULD be performed on the LUN that is identified by the disk index in the **OrigWTD** data element (section 3.1.1.5). If that is not successful, the server SHOULD return ERROR_FILE_NOT_FOUND.
- Ensures the device is in a state that permits the rollback. If it is not available for exclusive access, or a rollback is already in process, the server SHOULD return an appropriate error.
- Starts the rollback process.
- Updates the value of the **Status** data element (section 3.1.1.5) to set the flag **SnapshotStatusRollbackInProgress**.

Note This method can return prior to completion of the rollback.

3.36.4.15 AbortRollback (Opnum 17)

The **AbortRollback** method stops a snapshot rollback that was started with the **Rollback** method (section 3.36.4.14).

```
HRESULT AbortRollback();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	The LUN implicitly referenced by this snapshot is invalid.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]AbortRollback** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Gets access to the device. The rollback SHOULD be in process on the LUN that is identified by the disk index in the **OrigWTD** data element (section 3.1.1.5). If the LUN is invalid, this method SHOULD return E_INVALIDARG.
- Stops the rollback process.
- Updates the value of the **Status** data element (section 3.1.1.5) to clear the flag **SnapshotStatusRollbackInProgress**.

3.36.4.16 ExportAsWTDisk (Opnum 18)

The **ExportAsWTDisk** method is used to export the snapshot of a LUN as a VHD disk.

```
HRESULT ExportAsWTDisk(
        [out] unsigned long * pWTD,
        [in, out] VDS_LUN_INFORMATION * pVdsLunInfo);
```

pWTD: A pointer to a value that returns the disk index of the exported snapshot.

pVdsLunInfo: A pointer to a **VDS_LUN_INFORMATION** structure (section 2.2) that returns information about the VHD disk LUN for the exported snapshot.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY	The device is not ready.

Return value/code	Description
0x80070015	
E_ACCESSDENIED 0x80070005	General access denied error.
E_ABORT 0x80004004	The current snapshot has already been exposed as a LUN.
ERROR_BUSY 0x800700aa	The same snapshot is currently being exposed as a LUN.
E_UNEXPECTED 0x8000FFFF	The VSS snapshot shadow copy set is in an invalid state.
E_INVALIDARG 0x80070057	One or more arguments are invalid.
ERROR_FILE_NOT_FOUND 0x80070002	The system cannot find the file specified.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]ExportAsWTDisk** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Ensures the snapshot is suitable for exporting. If the snapshot has already been exported, or another export operation is currently running, the server SHOULD return an appropriate error.
- Derives the device path to this snapshot using the disk index defined in the **OrigWTD** data element (section 3.1.1.5). If the device cannot be found, the server SHOULD return E_INVALIDARG.
- Determines the type of device to create, based on the **Type** data element (section 3.1.1.4) associated with the snapshot device.
- Creates the disk to export the snapshot; see the server disk manager **IWTDiskMgr** interface (section 3.46) for details.
- Saves the disk index of the LUN that is associated with the exported snapshot in the **ExportedWTD** data element (section 3.1.1.5).
- Saves the current time in the **TimeStamp** data element (section 3.1.1.5).
- Updates the value of the **Status** data element (section 3.1.1.5) to set the flag **SnapshotStatusExported**.
- Returns the disk index of the exported snapshot by using the *pWTD* parameter.
- Return the information about the VHD disk LUN in a **VDS_LUN_INFORMATION** structure by using the *pVdsLunInfo* parameter.

3.36.4.17 GetVdsLunInfo (Opnum 19)

The **GetVdsLunInfo** method returns information about the VHD disk LUN for the exported snapshot.

```
HRESULT GetVdsLunInfo(
        [in, out] VDS_LUN_INFORMATION * pVdsLunInfo);
```

pVdsLunInfo: A pointer to a **VDS_LUN_INFORMATION** structure (section 2.2) that returns information about the VHD disk LUN for the exported snapshot.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_FAIL 0x80004005	The snapshot has not been previously exposed as a LUN.
ERROR_FILE_NOT_FOUND 0x80070002	The LUN exposed from this snapshot is not valid.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]GetVdsLunInfo** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Ensures the snapshot has been exported by checking the **SnapshotStatusExported** flag in the **Status** data element (section 3.1.1.5). If the snapshot has not been exported, the server SHOULD return E_FAIL.
- Finds the exported disk for this exported snapshot using the disk index defined in the **ExportedWTD** data element (section 3.1.1.5). If the disk cannot be found, the server SHOULD return ERROR_FILE_NOT_FOUND.
- Returns the information about the VHD disk LUN in a **VDS_LUN_INFORMATION** structure by using the *pVdsLunInfo* parameter.

The **VDS_LUN_INFORMATION** structure SHOULD be initialized with an **m_deviceIdDescriptor** member **VDS_STORAGE_DEVICE_ID_DESCRIPTOR** structure (section 2.2) that specifies a set of two **VDS_STORAGE_IDENTIFIER** structures (section 2.2). The **VDS** storage identifier values that are used in the initialization are defined in the **VDS_STORAGE_IDENTIFIER_CODE_SET** and **VDS_STORAGE_IDENTIFIER_TYPE** enumerations (section 2.2).

The first structure SHOULD be initialized as follows:

- **m_CodeSet** set to **VDSStorageIdCodeSetBinary**.
- **m_Type** set to **VDSStorageIdTypeFCPHName**.
- **m_rgbIdentifier** set to a **LOGICAL_UNIT_IDENTIFIER** structure (section 2.2.4.9).

The second structure SHOULD be initialized as follows:

- **m_CodeSet** set to **VDSStorageIdCodeSetBinary**.
- **m_Type** set to **VDSStorageIdTypeVendorSpecific**.
- **m_rgbIdentifier** set to a **WINTARGET_DISK_DEVICE_IDENTIFIER** structure (section 2.2.4.21). The **SnapshotId** member of the **WINTARGET_DISK_DEVICE_IDENTIFIER** structure SHOULD be initialized to the value of the **Id** data element (section 3.1.1.5) of the snapshot.

The **VDS_LUN_INFORMATION** structure SHOULD also be initialized with an **m_BusType** member **VDS_STORAGE_BUS_TYPE** value (section 2.2) of **VDSBusTypeScsi**.

3.36.4.18 DVMount (Opnum 20)

The **DVMount** method is used to surface the snapshot for read-only access on the iSCSI Software Target computer.

```
HRESULT DVMount();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	The LUN exposed from this snapshot is not valid or of an invalid type.
E_FAIL 0x80004005	The LUN exposed through this snapshot is disabled.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]DVMount** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Derives the device path to this snapshot using the disk index defined in the **OrigWTD** data element (section 3.1.1.5).
- Ensures the VHD disk associated with the snapshot is not disabled; otherwise, the server SHOULD return E_FAIL.
- Ensures the storage type of the device underlying the VHD disk of the snapshot is file-based, by checking the **Type** data element (section 3.1.1.4); otherwise, the server SHOULD return E_INVALIDARG.
- Surfaces the disk for read-only access; see the server disk **IWTDisk** interface (section 3.43) for details.
- Updates the value of the **Status** data element (section 3.1.1.5) to set the **SnapshotStatusLocallyMounted** flag.

3.36.4.19 DVDismount (Opnum 21)

The **DVDismount** method is used to revert the action performed by the **DVMount** method (section 3.36.4.18).

```
HRESULT DVDismount();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_FILE_NOT_FOUND 0x80070002	The LUN exposed from this snapshot is not valid.
E_INVALIDARG 0x80070057	The LUN exposed from this snapshot is not of the appropriate type.
ERROR_INVALID_FUNCTION 0x80070001	In an HA cluster configuration, the operation is not supported while the resource is not online. Alternatively, the type of the exposed LUN is invalid for the operation.

Return value/code	Description
ERROR_INVALID_OPERATION 0x800710dd	The type of the exposed LUN is invalid for the operation.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]DVDismount** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

Only the snapshot that mounted the disk can dismount it.

The server SHOULD process this method as follows:

- Derives the device path to this snapshot using the disk index defined in the **OrigWTD** data element (section 3.1.1.5).
- Dismounst the disk; see the server disk **IWTDisk** interface (section 3.43) for details.
- Updates the value of the **Status** data element (section 3.1.1.5) to clear the **SnapshotStatusLocallyMounted** flag.

3.36.5 Timer Events

None.

3.36.6 Other Local Events

None.

3.37 ISnapshot Client Details

The **ISnapshot** interface (section 3.36) enables ISTM to manage a particular snapshot.

3.37.1 Abstract Data Model

The client data model is specified in section 3.2.1.

3.37.2 Timers

None.

3.37.3 Initialization

Client initialization is specified in section 3.2.3.

3.37.4 Message Processing Events and Sequencing Rules

This section specifies client processing of error returns by methods of the **ISnapshot** interface (section 3.36).

3.37.4.1 Rollback (Opnum 16)

Processing of the following error return SHOULD be performed by client callers of this server method:

ERROR_SHARING_VIOLATION

A return value of ERROR_SHARING_VIOLATION indicates that the LUN associated with the snapshot is currently in use by another process. Some of the conditions that cause this error are transient, and the operation can be retried at a later time.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the rollback operation failed and SHOULD identify some of the possible causes of the failure.<79>

3.37.5 Timer Events

None.

3.37.6 Other Local Events

None.

3.38 ISnapshotMgr Server Details

The **ISnapshotMgr** interface is used to manage disk backup and restore operations.

3.38.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the snapshot management data model (section 3.1.1.5).

3.38.2 Timers

None.

3.38.3 Initialization

Snapshot manager initialization is specified in section 3.1.3

3.38.4 Message Processing Events and Sequencing Rules

The **ISnapshotMgr** interface includes the following methods:

Method	Description
EnumSnapshots	Returns an enumeration of all snapshots defined on the iSCSI Software Target. Opnum: 3
GetSnapshot	Returns a specified existing snapshot. Opnum: 4
PrepareCreateSnapshot	Prepares for the creation of a snapshot of a specified LUN. Opnum: 5

Method	Description
DoCreateSnapshot	Starts the process of performing a shadow copy snapshot of a LUN. Opnum: 6
DoEndSnapshot	Completes the process of performing a shadow copy snapshot of a LUN. Opnum: 7
DeleteSnapshot	Deletes a specified snapshot. Opnum: 8
ImportSnapshot	Adds the specified snapshot to the in-memory collection of existing snapshots. Opnum: 9

3.38.4.1 EnumSnapshots (Opnum 3)

The **EnumSnapshots** method returns an enumeration of all snapshots defined on the iSCSI Software Target.

```
HRESULT EnumSnapshots(
        [out] IEnumSnapshot ** ppEnumSnapshot);
```

ppEnumSnapshot: A pointer to an **IEnumSnapshot** interface (section 3.15) pointer that returns the snapshot enumeration. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumSnapshots** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates or updates the collection of snapshots that exist on the iSCSI Software Target and saves the result in the **SnapshotList** data object (section 3.1.1.5). This SHOULD include snapshots that were created outside ISTM. For example, a volume shadow copy service (VSS), if available on the operating system running ISTM, could have been used by another application to create snapshots of LUNs.

- Creates an **IEnumSnapshot** interface object and associates with it the collection of snapshots.
- Returns the **IEnumSnapshot** interface pointer by using the *ppEnumSnapshot* parameter.

3.38.4.2 GetSnapshot (Opnum 4)

The **GetSnapshot** method returns a specified existing snapshot.

```
HRESULT GetSnapshot(
        [in] GUID SnapshotId,
        [out] ISnapshot ** ppSnapshot);
```

SnapshotId: A **GUID** structure (section 2.2) that identifies the snapshot to return.

ppSnapshot: A pointer to an **ISnapshot** interface (section 3.36) pointer that returns the snapshot. This parameter MUST NOT be NULL

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_FILE_NOT_FOUND 0x80070002	The specified snapshot identifier was not found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **GetSnapshot** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Uses the specified snapshot identifier to get the **ISnapshot** interface object that is associated with the snapshot. If the snapshot instance cannot be found, the server SHOULD return ERROR_FILE_NOT_FOUND.
- Returns the **ISnapshot** interface pointer by using the *ppSnapshot* parameter.

3.38.4.3 PrepareCreateSnapshot (Opnum 5)

The **PrepareCreateSnapshot** method prepares for the creation of a snapshot of a specified LUN.

```
HRESULT PrepareCreateSnapshot(
        [in] unsigned long nWTD,
        [in, out] GUID * pSnapshotId);
```

nWTD: The disk index of the device that is associated with the LUN for which a snapshot will be created.

pSnapshotId: A pointer to a **GUID** structure (section 2.2) that identifies the created snapshot. If the GUID is zero, an identifier SHOULD be generated and returned using this parameter.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_FILE_NOT_FOUND 0x80070002	The LUN identifier specified was not found or was disabled.
ERROR_INVALID_FUNCTION 0x80070001	The LUN associated with snapshot is of an invalid type for the requested operation.
E_INVALIDARG 0x80070057	The specified snapshot identifier was not found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **PrepareCreateSnapshot** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Uses the specified disk index to access the virtual disk that is associated with this request. If a problem is encountered in obtaining this access, the server SHOULD return an appropriate error.
- Checks whether the virtual disk is enabled; if it is not, the server SHOULD return ERROR_FILE_NOT_FOUND.
- Checks the **VhdType** property (section 2.2.3.9) of the LUN to determine whether snapshots are supported. If not, the server SHOULD return ERROR_INVALID_FUNCTION.<80>
- If a snapshot identifier is specified by using the *pSnapshotId* parameter, checks the GUID against the collection of known snapshots in the **SnapshotList** data object (section 3.1.1.5). The specified GUID MUST NOT match an existing snapshot identifier.
- Creates and initializes an **ISnapshot** interface object (section 3.36).

- Adds the **ISnapshot** interface object to the collection maintained in **SnapshotList**.
- Returns the snapshot identifier by using the *pSnapshot* parameter.

The creation of the actual shadow copy that comprises the snapshot is launched by calling the **DoCreateSnapshot** method (section 3.38.4.4).

3.38.4.4 DoCreateSnapshot (Opnum 6)

The **DoCreateSnapshot** method starts the process of performing a shadow copy snapshot of a LUN.

```
HRESULT DoCreateSnapshot(
        [in] GUID SnapshotId);
```

SnapshotId: A **GUID** structure (section 2.2) that identifies a snapshot that was initialized by the **PrepareCreateSnapshot** method (section 3.38.4.3).

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_FILE_NOT_FOUND 0x80070002	The specified snapshot identifier was not found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **DoCreateSnapshot** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Uses the specified GUID to find the snapshot in the collection of existing snapshots in the **SnapshotList** data object (section 3.1.1.5). If the snapshot is not in the collection, the server SHOULD return ERROR_FILE_NOT_FOUND.
- Start the process of making the shadow copy for the snapshot.

This method does not wait for the snapshot operation to complete. The **DoEndSnapshot** method (section 3.38.4.5) SHOULD be called to wait for snapshot completion.

3.38.4.5 DoEndSnapshot (Opnum 7)

The **DoEndSnapshot** method completes the process of performing a shadow copy snapshot of a LUN.

```
HRESULT DoEndSnapshot(
        [in] GUID SnapshotId);
```

SnapshotId: A **GUID** structure (section 2.2) that identifies a snapshot that was started by the **DoCreateSnapshot** method (section 3.38.4.4).

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_FILE_NOT_FOUND 0x80070002	The specified snapshot identifier was not found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) defined in section 2.2.1 or [MS-ERREF].

When the **DoEndSnapshot** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Uses the specified GUID to find the snapshot in the collection of existing snapshots in the **SnapshotList** data object (section 3.1.1.5). If the snapshot is not in the collection, the server SHOULD return ERROR_FILE_NOT_FOUND.
- Waits for completion of the process of making the shadow copy for the snapshot.

3.38.4.6 DeleteSnapshot (Opnum 8)

The **DeleteSnapshot** method deletes a specified snapshot.

```
HRESULT DeleteSnapshot(
        [in] GUID SnapshotId);
```

SnapshotId: A **GUID** structure (section 2.2) that identifies a snapshot.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_SHARING_VIOLATION 0x80070020	The LUN associated with the specified snapshot has pending statuses/operations.
ERROR_DRIVE_LOCKED 0x8007006c	The operation might be disallowed by the HA cluster configuration.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) defined in section 2.2.1 or [MS-ERREF].

When the **DeleteSnapshot** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Uses the specified GUID to find the snapshot in the collection of existing snapshots in the **SnapshotList** data object (section 3.1.1.5). If the snapshot is not in the collection, the server SHOULD return S_OK.
- Checks whether there are pending operations involving the snapshot that would prevent its deletion. For example, if the snapshot is locally mounted, or if a rollback is in process, the server SHOULD return ERROR_SHARING_VIOLATION.
- Checks whether the iSCSI Software Target is in an HA cluster configuration. If it is, the server SHOULD return ERROR_DRIVE_LOCKED.
- Checks whether the snapshot is exported as a LUN. If it is, the server MUST delete the exported LUN.
- Releases all resources associated with the snapshot.
- Removes the snapshot from the **SnapshotList** data object.

3.38.4.7 ImportSnapshot (Opnum 9)

The **ImportSnapshot** method MAY be used to add the specified snapshot to the in-memory collection of existing snapshots.<81>

```
HRESULT ImportSnapshot(
        [in] ISnapshot * pSnapshot);
```

pSnapshot: A pointer to an **ISnapshot** interface (section 3.36) that defines the snapshot to add.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

The specified snapshot is added to the collection of existing snapshots in the **SnapshotList** data object (section 3.1.1.5).

3.38.5 Timer Events

None.

3.38.6 Other Local Events

None.

3.39 ISnapshotMgr Client Details

The **ISnapshotMgr** interface (section 3.38) is used to manage disk backup and restore operations.

3.39.1 Abstract Data Model

The client data model is specified in section 3.2.1.

3.39.2 Timers

None.

3.39.3 Initialization

Client initialization is specified in section 3.2.3.

3.39.4 Message Processing Events and Sequencing Rules

This section specifies client processing of error returns by methods of the **ISnapshotMgr** interface (section 3.38).

3.39.4.1 DeleteSnapshot (Opnum 8)

Processing of the following error returns SHOULD be performed by client callers of this server method:

ERROR_SHARING_VIOLATION

A return value of ERROR_SHARING_VIOLATION indicates that the LUN associated with the snapshot is currently in use by another process. Some of the conditions that cause this error are transient, and the operation can be retried at a later time.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the snapshot deletion failed and SHOULD identify some of the possible causes of the failure.<82>

ERROR_DRIVE_LOCKED

A return value of ERROR_DRIVE_LOCKED indicates that the LUN associated with the snapshot is currently in use by another process. Some of the conditions that cause this error are transient, and the operation can be retried at a later time.

This return value can also occur on an attempt to delete a snapshot in a HA cluster configuration.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the snapshot deletion failed and SHOULD identify some of the possible causes of the failure.<83>

3.39.5 Timer Events

None.

3.39.6 Other Local Events

None.

3.40 ISnsMgr Server Details

The **ISnsMgr** interface is used to manage information about iSNS servers.

3.40.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the iSNS management data model (section 3.1.1.6).

3.40.2 Timers

None.

3.40.3 Initialization

A collection of iSNS servers that exist on the iSCSI Software Target computer SHOULD be created in the **SnsServerList** data object (section 3.1.1.6).<84> Each element in the collection SHOULD be an instance of **SNS_SERVER** structure (section 2.2.4.18).

A collection of iSCSI initiators that have logged onto targets on iSNS servers MAY be created in the **CachedInitiatorList** data object (section 3.1.1.6).<85> Each element in the collection SHOULD be an instance of **CACHED_INITIATOR** structure (section 2.2.4.1).

Additional iSNS manager initialization is specified in section 3.1.3.

3.40.4 Message Processing Events and Sequencing Rules

The **ISnsMgr** interface includes the following methods:

Method	Description
EnumSnsServers	Returns an enumeration of all iSNS servers configured for the iSCSI Software Target. Opnum: 3
AddSnsServer	Adds a specified iSNS server to the collection. Opnum: 4
RemoveSnsServer	Removes a specified iSNS server from the collection. Opnum: 5
EnumCachedInitiators	Returns an enumeration of all cached initiators defined on the iSCSI Software Target. Opnum: 6

3.40.4.1 EnumSnsServers (Opnum 3)

The **EnumSnsServers** method returns an enumeration of all iSNS servers configured for the iSCSI Software Target.

```
HRESULT EnumSnsServers(
        [out] IEnumSnsServer ** ppEnumSnsServer);
```

ppEnumSnsServer: A pointer to an **IEnumSnsServer** interface pointer (section 3.16) that returns the enumeration of iSNS servers. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumSnsServers** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates an **IEnumSnsServer** interface object and associates with it the collection of iSNS servers that exist on the iSCSI Software Target computer that is maintained in the **SnsServerList** data object (section 3.1.1.6). Each element in the collection SHOULD be an instance of **SNS_SERVER** structure (section 2.2.4.18).

- Returns the **IEnumSnsServer** interface pointer by using the *ppEnumSnsServer* parameter.

3.40.4.2 AddSnsServer (Opnum 4)

The **AddSnsServer** method adds a specified iSNS server to the collection of known iSNS servers that is maintained by the iSNS manager.

```
HRESULT AddSnsServer(
        [in] BSTR SnsServerName);
```

SnsServerName: The name of the iSNS server.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **AddSnsServer** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Checks whether the specified iSNS server is already in the collection maintained in the **SnsServerList** data object (section 3.1.1.6). If it is, the server SHOULD return S_OK.
- Adds the specified iSNS server to the collection.<86> The element added to the collection SHOULD be an instance of **SNS_SERVER** structure (section 2.2.4.18).
- Registers each target that exists on the server with the iSNS server. In an HA cluster configuration, all portals that have access to a target SHOULD be included in the target registration.

3.40.4.3 RemoveSnsServer (Opnum 5)

The **RemoveSnsServer** method removes a specified iSNS server from the collection of known iSNS servers that is maintained by the iSNS manager.

```
HRESULT RemoveSnsServer(
        [in] BSTR SnsServerName);
```

SnsServerName: A BSTR, defined in [MS-OAUT].

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_FILE_NOT_FOUND 0x80070002	The supplied iSNS server address is not found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **AddSnsServer** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Checks whether the specified iSNS server is in the collection maintained in the **SnsServerList** data object (section 3.1.1.6). If it is not, the server SHOULD return ERROR_FILE_NOT_FOUND.
- Removes the specified iSNS server from the collection.<87>

3.40.4.4 EnumCachedInitiators (Opnum 6)

The **EnumCachedInitiators** method returns an enumeration of all cached iSCSI initiators that have logged onto targets on iSNS servers.

```
HRESULT EnumCachedInitiators(
        [out] IEnumCachedInitiator ** ppEnumCachedInitiator);
```

ppEnumCachedInitiator: A pointer to an **IEnumCachedInitiator** interface pointer (section 3.4) that returns the cached initiator enumeration. This parameter MUST NOT be NULL

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY	The device is not ready.

Return value/code	Description
0x80070015	
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumCachedInitiators** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates an **IEnumCachedInitiator** interface object and associates with it the collection of initiators that have logged onto targets on iSNS servers that MAY be maintained in the **CachedInitiatorList** data object (section 3.1.1.6). <88> Each element in the collection SHOULD be an instance of **CACHED_INITIATOR** structure (section 2.2.4.1).
- Returns the **IEnumCachedInitator** interface pointer by using the *ppEnumCachedInitiator* parameter.

3.40.5 Timer Events

None.

3.40.6 Other Local Events

None.

3.41 IStatusNotify Server Details

The **IStatusNotify** interface allows the ISTM server to send status notifications to ISTM clients. An ISTM client implements this interface and performs the role of server. The ISTM server performs the role of client of this interface, calling the methods and handling the return values.

3.41.1 Abstract Data Model

A data model is maintained by the ISTM client to support status notifications. It includes the following persisted data:

StatusNotify: An instance of the **IStatusNotify** interface (section 3.41.4) that defines methods that are called on the occurrence of notification events. A pointer to this interface is passed to the ISTM server when registering and unregistering for status notifications.

Cookie: A unique identifier for callback registration. The value of the cookie is passed to the ISTM server when registering for status notifications, and the ISTM server passes it back in notification method invocations.

The **StatusNotify**/**Cookie** pair allows an ISTM client to have multiple callback registrations for the same instance of the **IStatusNotify** interface.

3.41.2 Timers

None.

3.41.3 Initialization

An ISTM client initializes status notifications by performing the following steps:

Obtain a pointer to the **IWTGeneral** interface (section 3.52) by performing DCOM activation on the CLSID of that interface (section 1.9).

Create a **StatusNotify** data object (section 3.41.1).

Invoke the **RegisterNotification** method (section 3.52.4.9) on the **IWTGeneral** interface, passing in the **StatusNotify** object. The ISTM client calls **UnregisterNotification** (section 3.52.4.10) to direct the ISTM server not to perform further status notifications.

3.41.4 Message Processing Events and Sequencing Rules

The **IStatusNotify** interface includes the following methods.

Method	Description
LunListChanged	Performs an implementation-defined action. Opnum: 3
LunStatusUpdate	Performs an implementation-defined action. Opnum: 4

3.41.4.1 LunListChanged (Opnum 3)

The **LunListChanged** method performs an implementation-defined action. It is called by the ISTM server to notify the ISTM client that a change has occurred in the mapping of LUNs to targets.

```
HRESULT LunListChanged(
        [in] unsigned long lCookie);
```

lCookie: A value defined by the ISTM client that identifies the instance of registered notifications.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

See **LunListChanged** (section 3.42.4.1) for a description of events that can cause a change in the LUN list.

3.41.4.2 LunStatusUpdate (Opnum 4)

The **LunStatusUpdate** method performs an implementation-defined action. It is called by the ISTM server to notify the ISTM client that a change has occurred in the status of LUNs mapped to targets.

```
HRESULT LunStatusUpdate(
        [in] unsigned long lCookie,
        [in] LUN_STATUS * pLunStatus);
```

lCookie: A value defined by the ISTM client that identifies the instance of registered notifications.

pLunStatus: A pointer to a **LUN_STATUS** structure (section 2.2.4.11) that describes the change in LUN status that has occurred.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

See **LunStatusUpdate** (section 3.42.4.2) for a description of events that can cause a LUN status update.

3.41.5 Timer Events

None.

3.41.6 Other Local Events

None.

3.42 IStatusNotify Client Details

The **IStatusNotify** interface allows the ISTM server to send status notifications to ISTM clients. An ISTM client implements this interface and performs the role of server. The ISTM server performs the role of client of this interface, calling the methods and handling the return values.

3.42.1 Abstract Data Model

A data model is maintained by the ISTM server to support status notifications. It includes the following persisted data:

StatusNotifyList: A collection of callback-context-identifier elements that contain the data for all ISTM clients that have registered for LUN status notifications.

StatusNotify: The callback-context-identifier element that an ISTM client has registered for status notifications. Logically, this context is uniquely associated with the **IStatusNotify** interface (section 3.41) pointer created by the ISTM client.

Cookie: A unique identifier for callback registration. The value of the cookie is managed by the ISTM client. It is passed to the ISTM server when the client registers for notifications, and it is passed back to the client on notification method invocations.

The **StatusNotify/Cookie** pair allows an ISTM client to have multiple callback registrations for the same instance of the **IStatusNotify** interface.

ProgressTrackingInformation: The progress tracking information is maintained for each LUN that has ongoing restore operations. It contains the current status of a restore operation being performed on a LUN. This structure is passed to an ISTM client that has registered for status notifications. It includes the following information:

The current device status.

The completion percentage of a restore operation.

The completion code returned from a restore operation.

3.42.2 Timers

None.

3.42.3 Initialization

Feature manager initialization is specified in section 3.1.3.

3.42.4 Message Processing Events and Sequencing Rules

This section specifies ISTM server processing in the client role for status notifications.

3.42.4.1 LunListChanged (Opnum 3)

The ISTM server performs this notification if a change occurs in LUN definitions, including:

- A LUN assignment to an iSCSI target is added.
- A LUN assignment to an iSCSI target is removed.
- An iSCSI target started using a LUN.
- An iSCSI target stopped using a LUN.
- A LUN is created.
- A LUN is deleted.
- A disk associated with a LUN is enabled.
- A disk associated with a LUN is disabled.

Any error returned by the implementation of **ListLunChanged** by the ISTM client will cause the server to stop performing further status notification on the interface.

3.42.4.2 LunStatusUpdate (Opnum 4)

The ISTM server performs this notification if a change occurs in LUN status, including:

- The status changes of the device associated with a LUN.
- The completion percentage changes of a snapshot restore on a device associated with a LUN.

- A snapshot restore completes on a device associated with a LUN.

Any error returned by the implementation of **ListLunChanged** by the ISTM client will cause the server to stop performing further status notification on the interface.

3.42.5 Timer Events

None.

3.42.6 Other Local Events

None.

3.43 IWTDisk Server Details

The **IWTDisk** interface is used to configure and maintain individual virtual disks.

3.43.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the LUN management data model (section 3.1.1.4).

3.43.2 Timers

None.

3.43.3 Initialization

LUN manager initialization is specified in section 3.1.3.

3.43.4 Message Processing Events and Sequencing Rules

The **IWTDisk** interface includes the following methods:

Method	Description
[propget]Guid	Returns the unique identifier of the virtual disk. Opnum: 3
[propget]SerialNumber	Returns the serial number of the virtual disk. Opnum: 4
[propget]WTD	Returns the index of the LUN associated with the virtual disk. Opnum: 5
[propput]WTD	Sets the index of the LUN associated with the virtual disk. Opnum: 6
[propget]Type	Returns the type of the virtual disk. Opnum: 7
[propput]Type	Sets the type of the virtual disk. Opnum: 8

Method	Description
[propget]Flags	Returns the mapping flags of the virtual disk. Opnum: 9
[propput]Flags	Sets the mapping flags of the virtual disk. Opnum: 10
[propget]Status	Returns the device status of the virtual disk. Opnum: 11
[propput]Status	Sets the device status of the virtual disk. Opnum: 12
[propget]Assigned	Returns whether the virtual disk is assigned to an iSCSI target. Opnum: 13
[propput]Assigned	Sets whether the virtual disk is assigned to an iSCSI target. Opnum: 14
[propget]Description	Returns the user-friendly description of the virtual disk. Opnum: 15
[propput]Description	Sets the user-friendly description of the virtual disk. Opnum: 16
[propget]DevicePath	Returns the path to the virtual disk. Opnum: 17
[propput]DevicePath	Sets the path to the virtual disk. Opnum: 18
[propget]Size	Returns the size of the virtual disk. Opnum: 19
[propget]SnapshotStoragePath	Returns the backup storage path of the virtual disk. Opnum: 20
[propput]SnapshotStoragePath	Sets the backup storage path of the virtual disk. Opnum: 21
[propget]SnapshotStorageSize	Returns the size of the space available for backup of the virtual disk. Opnum: 22
[propput]SnapshotStorageSize	Sets the size of the space available for backup of the virtual disk. Opnum: 23
[propget]SnapshotStorageUsedSize	Returns the size of the space currently used for backup of the virtual disk. Opnum: 24
[propget]LMType	Returns the local mount type of the virtual disk. Opnum: 25
[propput]LMType	Sets the local mount type of the virtual disk.

Method	Description
	Opnum: 26
[propget]LMStatus	Returns the local mount status of the virtual disk. Opnum: 27
[propget]LMTimeStamp	Returns the local mount time stamp of the virtual disk. Opnum: 28
[propget]LMWTManageMountPoint	Returns whether the virtual disk is admissible for local mount operations. Opnum: 29
[propput]LMWTManageMountPoint	Sets whether the virtual disk is admissible for local mount operations. Opnum: 30
[propget]LMMountPoint	Returns the local mount point of the virtual disk. Opnum: 31
[propput]LMMountPoint	Sets the local mount point of the virtual disk. Opnum: 32
[propget]LMCurrMountDeviceId	Returns the plug and play identifier of the virtual disk. Opnum: 33
[propput]LMCurrMountDeviceId	Sets the plug and play identifier of the virtual disk. Opnum: 34
[propget]LMSnapshotId	Returns the local mount snapshot identifier of the virtual disk. Opnum: 35
Save	Saves virtual disk settings to persistent storage. Opnum: 36
DVMountDirect	Mounts the virtual disk locally for read and write access. Opnum: 37
DVDismount	Dismounts the locally mounted virtual disk. Opnum: 38
EnumLMMountPoints	Returns an enumeration of the mount points associated with the virtual disk. Opnum: 39
GetVdsLunInfo	Retrieves the LUN information associated with the virtual disk. Opnum: 40
GetDeviceVolumeGuid	Retrieves the GUID of the volume on which the virtual disk resides. Opnum: 41

3.43.4.1 [propget]Guid (Opnum 3)

The **[propget]Guid** method returns the unique identifier of the virtual disk.

```
[propget] HRESULT Guid(
        [out] [retval] GUID * pGuid);
```

pGuid: A pointer to the **GUID** structure (section 2.2) that returns the unique identifier. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server obtains the unique identifier of the virtual disk from the **Guid** data element (section 3.1.1.4), and returns it by using the *pGuid* parameter.

3.43.4.2 [propget]SerialNumber (Opnum 4)

The **[propget]SerialNumber** method returns the serial number of the virtual disk.

```
[propget] HRESULT SerialNumber(
        [out] [retval] BSTR * pSerialNumber);
```

pSerialNumber: A pointer to the string that returns the serial number. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.
E_POINTER 0x80004003	Invalid pointer.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. If the *pSerialNumber* parameter is NULL or if the virtual disk was not properly initialized, the server returns E_POINTER. Otherwise, the server obtains the serial number of the virtual disk from the **SerialNumber** data element (section 3.1.1.4), and returns it by using the *pSerialNumber* parameter.

3.43.4.3 [propget]WTD (Opnum 5)

The **[propget]WTD** method returns the index of the LUN associated with the virtual disk.

```
[propget] HRESULT WTD(
        [out] [retval] unsigned long * pWTD);
```

pWTD: A pointer to an unsigned value that returns the index. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

When the **[propget]WTD** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server then obtains the index of the LUN associated with the virtual disk from the **WTDIndex** data element (section 3.1.1.4), and returns it by using the *pWTD* parameter.

3.43.4.4 [propput]WTD (Opnum 6)

The **[propput]WTD** method MAY set the index of the LUN associated with the virtual disk.<89>

```
[propput] HRESULT WTD(
        [in] unsigned long nWTD);
```

nWTD: The index of the LUN associated with the virtual disk.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.5 [propget]Type (Opnum 7)

The **[propget]Type** method returns the type of the virtual disk.

```
[propget] HRESULT Type(
        [out] [retval] WTDType * pType);
```

pType: A pointer to a **WTDType** value (section 2.2.3.10) that returns the type. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

When the **[propget]Type** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server obtains the type of the virtual disk from the **Type** data element (section 3.1.1.4), and returns it by using the *pType* parameter.

3.43.4.6 [propput]Type (Opnum 8)

The **[propput]Type** method MAY set the type of the virtual disk.<90>

```
[propput] HRESULT Type(
        [in] WTDType eType);
```

eType: The **WTDType** value (section 2.2.3.10) that specifies the type.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.7 [propget]Flags (Opnum 9)

The **[propget]Flags** method returns the mapping flags of the virtual disk.

```
[propget] HRESULT Flags(
        [out] [retval] unsigned long * pFlags);
```

pFlags: A pointer to an unsigned value that returns the mapping flags. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

When the **[propget]Flags** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server obtains the mapping flags of the virtual disk from the **Flags** data element (section 3.1.1.4), and returns it by using the *pFlags* parameter.

3.43.4.8 [propput]Flags (Opnum 10)

The **[propput]Flags** method MAY set the mapping flags of the virtual disk.<91>

```
[propput] HRESULT Flags(
        [in] unsigned long nFlags);
```

nFlags: The mapping flags.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.9 [propget]Status (Opnum 11)

The **[propget]Status** method returns the device status of the virtual disk.

```
[propget] HRESULT Status(
        [out] [retval] DeviceStatus * pStatus);
```

pStatus: A pointer to the **DeviceStatus** value (section 2.2.3.3) that returns the device status. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

When the **[propget]Status** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server obtains the device status of the virtual disk from the **Status** data element (section 3.1.1.4), and returns it by using the *pStatus* parameter.

Note If the virtual disk has not been properly initialized, the server returns a device status of **Idle**.

3.43.4.10 [propput]Status (Opnum 12)

The **[propput]Status** method MAY set the device status of the virtual disk.<92>

```
[propput] HRESULT Status(
        [in] DeviceStatus eStatus);
```

eStatus: The **DeviceStatus** value (section 2.2.3.3) that specifies the device status.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.11 [propget]Assigned (Opnum 13)

The **[propget]Assigned** method MAY return whether the virtual disk is assigned to an iSCSI target.<93>

```
[propget] HRESULT Assigned(
        [out] [retval] boolean * pAssigned);
```

pAssigned: A pointer to a Boolean value that returns whether the virtual disk is assigned. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.12 [propput]Assigned (Opnum 14)

The **[propput]Assigned** method MAY set whether the virtual disk is assigned to an iSCSI target.<94>

```
[propput] HRESULT Assigned(
        [in] boolean bAssigned);
```

bAssigned: A Boolean value that specifies whether the virtual disk is assigned.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.13 [propget]Description (Opnum 15)

The **[propget]Description** method returns the user-friendly description of the virtual disk.

```
[propget] HRESULT Description(
          [out] [retval] BSTR * pDescription);
```

pDescription: A pointer to a string that returns the user-friendly description. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Description** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server obtains the user-friendly description of the virtual disk from the **Description** data element (section 3.1.1.4), and returns it by using the *pDescription* parameter.

3.43.4.14 [propput]Description (Opnum 16)

The **[propput]Description** method sets the user-friendly description of the virtual disk.

```
[propput] HRESULT Description(
        [in] BSTR Description);
```

Description: A string that specifies the user-friendly description.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]Description** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server sets the user-friendly description of the virtual disk in the **Description** data element (section 3.1.1.4) to the string in the *Description* parameter.

3.43.4.15 [propget]DevicePath (Opnum 17)

The **[propget]DevicePath** method returns the path to the virtual disk.

```
[propget] HRESULT DevicePath(
        [out] [retval] BSTR * pDevicePath);
```

pDevicePath: A pointer to a string that returns the path. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

Return value/code	Description
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]DevicePath** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server then obtains the path to the virtual disk from the **DevicePath** data element (section 3.1.1.4), and returns it by using the *pDevicePath* parameter.

3.43.4.16 [propput]DevicePath (Opnum 18)

The **[propput]DevicePath** method sets the path to the virtual disk.

```
[propput] HRESULT DevicePath(
        [in] BSTR DevicePath);
```

DevicePath: A string that specifies the path.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]DevicePath** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server sets the path to the virtual disk in the **DevicePath** data element (section 3.1.1.4) to the string in the *DevicePath* parameter.

3.43.4.17 [propget]Size (Opnum 19)

The **[propget]Size** method returns the size of the virtual disk.

```
[propget] HRESULT Size(
        [out] [retval] hyper * pSize);
```

pSize: A pointer to a signed value that returns the size. The value that is returned SHOULD be interpreted as an unsigned value. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.
E_POINTER 0x80004003	Invalid pointer.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Size** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. The server also verifies that the *pSize* parameter is not NULL, returning E_POINTER if it is.

The server then obtains the size of the virtual disk from the **Size** data element (section 3.1.1.4), and returns it by using the *pSize* parameter.

Note If the virtual disk has not been properly initialized, the server returns a value of zero by using the *pSize* parameter.

3.43.4.18 [propget]SnapshotStoragePath (Opnum 20)

The **[propget]SnapshotStoragePath** method returns the backup storage path of the virtual disk.

```
[propget] HRESULT SnapshotStoragePath(
        [out] [retval] BSTR * pSnapshotStoragePath);
```

pSnapshotStoragePath: A pointer to a string that returns the backup storage path. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed, or the state of VHD is not comptible with Snapsthot operations
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.
E_INVALIDARG 0x80070057	The type of the LUN is incompatible with snapshot operations/configurations.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]SnapshotStoragePath** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

If the **Type** data element (section 3.1.1.4) of the virtual disk is not set to **WTDTypeFileBased** (section 2.2.3.10), the server SHOULD return E_INVALIDARG.

Under the following conditions, the server SHOULD return S_OK, but also SHOULD return an empty string by using the *pSnapshotStoragePath* parameter:

- The virtual disk is not enabled.
- The virtual disk is a shadow copy of another virtual disk.
- The **VhdType** data element (section 3.1.1.4) of the virtual disk is set to **DifferencingVhd** (section 2.2.3.9).

Otherwise, the server obtains the backup storage path of the virtual disk from the **SnapshotStoragePath** data element (section 3.1.1.4), and returns it by using the *pSnapshotStoragePath* parameter.

3.43.4.19 [propput]SnapshotStoragePath (Opnum 21)

The **[propput]SnapshotStoragePath** method sets the backup storage path of the virtual disk.

```
[propput] HRESULT SnapshotStoragePath(
        [in] BSTR SnapshotStoragePath);
```

SnapshotStoragePath: A string that specifies the backup storage path.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.
E_INVALIDARG 0x80070057	The type or the current state of the LUN is incompatible with snapshot operations/configurations.
VSS_E_VOLUME_NOT_SUPPORTED 0x8004230C	Shadow copying the specified volume is not supported.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) defined in section 2.2.1 or [MS-ERREF].

When the **[propput]SnapshotStoragePath** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

If the **Type** data element (section 3.1.1.4) of the virtual disk is not set to **WTDTypeFileBased** (section 2.2.3.10), or if the **VhdType** data element of the virtual disk is not set to **FixedVhd** (section 2.2.3.9), the server SHOULD return E_INVALIDARG.

If the iSCSI Software Target computer is part of an HA cluster, the server verifies that the specified backup storage path is compatible with the cluster constraints and resource relationship, including the following:

- The volume of the virtual disk and the backup storage path are on the same cluster disk

- The cluster disk is owned by the current node.
- The cluster disk is a basic disk.

If either of these tests fails, the server SHOULD return the value VSS_E_VOLUME_NOT_SUPPORTED (section 2.2.1). Otherwise, the server sets the backup storage path of the virtual disk to the string in the *SnapshotStoragePath* parameter by using the **SnapshotStoragePath** data element (section 3.1.1.4).

Note If the virtual disk has not been enabled, the server MUST NOT change its **SnapshotStoragePath** property, but simply return S_OK.

3.43.4.20 [propget]SnapshotStorageSize (Opnum 22)

The **[propget]SnapshotStorageSize** method returns the space available for backup of the virtual disk.

```
[propget] HRESULT SnapshotStorageSize(
        [out] [retval] hyper * pSize);
```

pSize: A pointer to a signed value that returns the space available for backup, in bytes. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation succeeded or the LUN does not support snapshot operations in the current state.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.
E_INVALIDARG 0x80070057	The type or the current type of LUN is incompatible with Snapshot operations/configurations.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) defined in section 2.2.1 or [MS-ERREF].

When the **[propget]SnapshotStorageSize** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

If the **Type** data element (section 3.1.1.4) of the virtual disk is not set to **WTDTypeFileBased** (section 2.2.3.10), the server SHOULD return E_INVALIDARG.

Under the following conditions, the server SHOULD return S_OK, but also SHOULD return a value of zero by using the *pSize* parameter:

- The virtual disk is not enabled.
- The virtual disk is a shadow copy of another virtual disk.
- The **VhdType** data element (section 3.1.1.4) of the virtual disk is set to **DifferencingVhd** (section 2.2.3.9).

Otherwise, the server obtains the space available for backup of the virtual disk from the **SnapshotStorageSize** data element (section 3.1.1.4), and returns it by using the *pSize* parameter.

3.43.4.21 [propput]SnapshotStorageSize (Opnum 23)

The **[propput]SnapshotStorageSize** method sets the space available for backup of the virtual disk.

```
[propput] HRESULT SnapshotStorageSize(
        [in] hyper nSize);
```

nSize: The space available for backup, in bytes.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.
E_INVALIDARG 0x80070057	The type or the current state of the LUN is incompatible with snapshot operations/configurations.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) defined in section 2.2.1 or [MS-ERREF].

When the **[propput]SnapshotStorageSize** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

The server SHOULD return E_INVALIDARG under the following conditions:

- If the **Type** data element (section 3.1.1.4) of the virtual disk is not set to **WTDTypeFileBased** (section 2.2.3.10).
- If the **VhdType** data element of the virtual disk is not set to **FixedVhd** (section 2.2.3.9).
- If the value in the *nSize* parameter is outside an acceptable range of values for the size of a snapshot store.<95>

Otherwise, the server SHOULD set the space available for backup of the virtual disk in the **SnapshotStorageSize** data element (section 3.1.1.4) to the value specified in the *nSize* parameter.

Note If the virtual disk has not been enabled, the server MUST NOT change its space available for backup, but SHOULD return S_OK.

3.43.4.22 [propget]SnapshotStorageUsedSize (Opnum 24)

The **[propget]SnapshotStorageUsedSize** method returns the space currently used for backup of the virtual disk.

```
[propget] HRESULT SnapshotStorageUsedSize(
        [out] [retval] hyper * pSize);
```

pSize: A pointer to a signed value that returns the space currently used for backup, in bytes. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation succeeded or the LUN does not support snapshot operations in the current state.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) defined in section 2.2.1 or [MS-ERREF].

When the **[propget]SnapshotStorageUsedSize** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

If the **Type** data element (section 3.1.1.4) of the virtual disk is not set to **WTDTypeFileBased** (section 2.2.3.10), the server SHOULD return E_INVALIDARG.

Under the following conditions, the server SHOULD return S_OK, but also SHOULD return a value of zero in the *pSize* parameter:

- The virtual disk is not enabled.
- The virtual disk is a shadow copy of another virtual disk.
- The **VhdType** data element (section 3.1.1.4) of the virtual disk is set to **DifferencingVhd** (section 2.2.3.9).

Otherwise, the server obtains the space currently used for backup of the virtual disk from the **SnapshotStorageUsedSize** data element (section 3.1.1.4), and returns it by using the *pSize* parameter.

3.43.4.23 [propget]LMType (Opnum 25)

The **[propget]LMType** method returns the local mount type of the virtual disk.

```
[propget] HRESULT LMType(
        [out] [retval] MountType * pType);
```

pType: A pointer to a **MountType** value (section 2.2.3.6) that returns the local mount type. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

When the **[propget]LMType** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server obtains the local mount type of the virtual disk from the **LMType** data element (section 3.1.1.4), and returns it by using the *pType* parameter.

3.43.4.24 [propput]LMType (Opnum 26)

The **[propput]LMType** method MAY set the local mount type of the virtual disk.<96>

```
[propput] HRESULT LMType(
        [in] MountType eType);
```

eType: A **MountType** value (section 2.2.3.6) that specifies the local mount type.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.25 [propget]LMStatus (Opnum 27)

The **[propget]LMStatus** method returns the local mount status of the virtual disk.

```
[propget] HRESULT LMStatus(
        [out] [retval] MountType * pStatus);
```

pStatus: A pointer to a **MountType** value (section 2.2.3.6) that returns the local mount status. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

Return value/code	Description
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.
E_INVALIDARG 0x80070057	The supplied pointer is invalid.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]LMStatus** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

The server then determines whether the virtual disk is mounted. If it is not mounted, the server returns **DoNotMount** by using the *pStatus* parameter. If it is mounted, the server determines whether it is a snapshot store, and returns **MountSnapshot** or **MountDirect**, as appropriate, by using the *pStatus* parameter.

3.43.4.26 [propget]LMTimeStamp (Opnum 28)

The **[propget]LMTimeStamp** method returns the local mount time stamp of the virtual disk.

```
[propget] HRESULT LMTimeStamp(
        [out] [retval] hyper * pTimeStamp);
```

pTimeStamp: A pointer to a value that returns the local mount time stamp. The value that is returned SHOULD be interpreted as a **FILETIME** structure (section 2.2). This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.
E_INVALIDARG	The supplied pointer is invalid.

Return value/code	Description
0x80070057	

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]LMTimeStamp** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server obtains the local mount time stamp of the virtual disk from the **LMTimeStamp** data element (section 3.1.1.4), and returns it by using the *pTimeStamp* parameter.

Note If there is no snapshot store associated with the virtual disk, the server SHOULD return S_OK and a value of zero by using the *pTimeStamp* parameter.

3.43.4.27 [propget]LMWTManageMountPoint (Opnum 29)

The **[propget]LMWTManageMountPoint** method MAY retrieve whether the virtual disk is admissible for local mount operations.<97>

```
[propget] HRESULT LMWTManageMountPoint(
        [out] [retval] boolean * pManage);
```

pManage: A pointer to a Boolean variable that returns whether the virtual disk is admissible for local mount operations. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.28 [propput]LMWTManageMountPoint (Opnum 30)

The **[propput]LMWTManageMountPoint** method MAY set whether the virtual disk is admissible for local mount operations.<98>

```
[propput] HRESULT LMWTManageMountPoint(
        [in] boolean bManage);
```

bManage: A Boolean value that specifies whether the virtual disk is admissible for local mount operations.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.29 [propget]LMMountPoint (Opnum 31)

The **[propget]LMMountPoint** method MAY return the local mount point of the virtual disk. <99>

```
[propget] HRESULT LMMountPoint(
        [out] [retval] BSTR * pPath);
```

pPath: A pointer to a string that returns the local mount point. This parameter MUST NOT be NULL.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.30 [propput]LMMountPoint (Opnum 32)

The **[propput]LMMountPoint** method MAY set the local mount point of the virtual disk.<100>

```
[propput] HRESULT LMMountPoint(
        [in] BSTR Path);
```

Path: A string that specifies the local mount point.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.31 [propget]LMCurrMountDeviceId (Opnum 33)

The **[propget]LMCurrMountDeviceId** method returns the plug and play identifier of the virtual disk.

```
[propget] HRESULT LMCurrMountDeviceId(
        [out] [retval] BSTR * pDeviceId);
```

pDeviceId: A pointer to a string that returns the plug and play identifier. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed or the LUN was not locally mounted.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

When the **[propget]LMCurrMountDeviceId** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server obtains the plug and play identifier of the virtual disk from the **LMCurrMountDeviceId** data element (section 3.1.1.4) and returns it by using the *pDeviceId* parameter.

3.43.4.32 [propput]LMCurrMountDeviceId (Opnum 34)

The **[propput]LMCurrMountDeviceId** method MAY set the plug and play identifier of the virtual disk.<101>

```
[propput] HRESULT LMCurrMountDeviceId(
        [in] BSTR DeviceId);
```

DeviceId: A string that specifies the plug and play identifier.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.43.4.33 [propget]LMSnapshotId (Opnum 35)

The **[propget]LMSnapshotId** method returns the local mount snapshot identifier of the virtual disk.

```
[propget] HRESULT LMSnapshotId(
        [out] [retval] GUID * pLMSnapshotId);
```

pLMSnapshotId: A pointer to a **GUID** structure (section 2.2) that returns the local mount snapshot identifier. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed or the LUN was not locally mounted.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]LMSnapshotId** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Verifies that the virtual disk is mounted locally.
- Obtains the local mount snapshot identifier of the virtual disk from the **LMSnapshotId** data element (section 3.1.1.4).
- Returns the local mount snapshot identifier by using the *pLMSnapshotId* parameter.

3.43.4.34 Save (Opnum 36)

The **Save** method saves virtual disk settings to persistent storage.

```
HRESULT Save();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

When the **Save** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD save to persistent storage all relevant virtual disk elements specified in the LUN management data model (section 3.1.1.4) and return S_OK.

3.43.4.35 DVMountDirect (Opnum 37)

The **DVMountDirect** method mounts the virtual disk locally for read and write access.

```
HRESULT DVMountDirect();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline, or the type of VHD supporting the LUN is incompatible with the operation.
E_INVALIDARG	The type of VHD supporting the LUN is incompatible with the operation.

Return value/code	Description
0x80070057	
E_FAIL 0x80004005	The status of the LUN is incompatible with the operation.
E_POINTER 0x80004003	The type of VHD supporting the LUN is incompatible with the operation.
ERROR_SHARING_VIOLATION 0x80070020	The current status of the LUN is incompatible with the operation.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **DVMountDirect** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD perform the following checks to ensure that local mounting of the virtual disk is appropriate:

- The virtual disk has been properly initialized; otherwise, the server SHOULD return E_POINTER.
- The virtual disk is enabled; otherwise, the server SHOULD return E_FAIL.
- The virtual disk is type **WTDTypeFileBased** (section 2.2.3.10) according to the **Type** data element (section 3.1.1.4) ; otherwise, the server SHOULD return E_INVALIDARG.
- The virtual disk is not type **DifferencingVhd** (section 2.2.3.9) according to the **VhdType** data element (section 3.1.1.4), and the virtual disk is not a shadow copy; otherwise, the server SHOULD return ERROR_INVALID_FUNCTION.
- The virtual disk is not in the process of being rolled back; otherwise, the server SHOULD return ERROR_SHARING_VIOLATION.

The server then dismounts the virtual disk if it is currently mounted using a process similar to that described for the **DVDismount** method (section 3.43.4.36), and then mounts it locally for read/write access using the relevant operating system directives.

3.43.4.36 DVDismount (Opnum 38)

The **DVDismount** method dismounts the locally mounted virtual disk.

```
HRESULT DVDismount();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline, or the type of VHD supporting the LUN is incompatible with the operation.
E_FAIL 0x80004005	The state of the LUN is incompatible with the operation.
E_POINTER 0x80004003	The type of VHD supporting the LUN is incompatible with the operation.
ERROR_INVALID_HANDLE 0x80070006	The local mount services are not available.
ERROR_INVALID_OPERATION 0x800710dd	The sequence of mount / dismount operation is incorrect.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **DVDismount** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD perform the following checks to ensure that dismounting the virtual disk is appropriate:

- The virtual disk has been properly initialized; otherwise, the server SHOULD return E_POINTER.
- The virtual disk is enabled; otherwise, the server SHOULD return E_FAIL.
- The virtual disk is not type **DifferencingVhd** (section 2.2.3.9) according to the **VhdType** data element (section 3.1.1.4) ; otherwise, the server SHOULD return ERROR_INVALID_FUNCTION.
- The virtual disk was not mounted from a snapshot store; otherwise, the server SHOULD return ERROR_INVALID_OPERATION.

The server then dismounts the virtual disk using the relevant operating system directives.

3.43.4.37 EnumLMMountPoints (Opnum 39)

The **EnumLMMountPoints** method returns an enumeration of the mount points associated with the virtual disk.

```
HRESULT EnumLMMountPoints(
        [out] IEnumLMMountPoint ** ppEnumLMMountPoint);
```

ppEnumLMMountPoint: A pointer to an **IEnumLMMountPoint** interface (section 3.10) pointer that returns the enumeration of mount points. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumLMMountPoints** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Creates a collection of information for all mount points that are associated with the virtual disk, with the elements of the collection defined according to the abstract data model (section 3.1.1.4) that is relevant to the **MOUNT_POINT** structure (section 2.2.4.121).
- Creates an **IEnumLMMountPoint** interface object and associates with it the collection of mount point information.
- Returns the **IEnumLMMountPoint** interface pointer by using the *ppEnumLMMountPoint* parameter.

3.43.4.38 GetVdsLunInfo (Opnum 40)

The **GetVdsLunInfo** method retrieves the SCSI information that is associated with the virtual disk.

```
HRESULT GetVdsLunInfo(
        [in, out] VDS_LUN_INFORMATION * pVdsLunInfo);
```

pVdsLunInfo: A pointer to a **VDS_LUN_INFORMATION** structure (section 2.2) that returns the SCSI information. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.
E_POINTER 0x80004003	The LUN was not properly initialized.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **GetVdsLunInfo** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- If the virtual disk has not been properly initialized, returns E_POINTER.
- Gathers the SCSI information about the virtual disk that is relevant to the **VDS_LUN_INFORMATION** structure.
- Returns the SCSI information in a **VDS_LUN_INFORMATION** structure by using the *pVdsLunInfo* parameter.

The **VDS_LUN_INFORMATION** SHOULD be initialized with an **m_deviceIdDescriptor** member **VDS_STORAGE_DEVICE_ID_DESCRIPTOR** structure (section 2.2) that specifies a set of two **VDS_STORAGE_IDENTIFIER** structures (section 2.2). The VDS storage identifier values that are used in the initialization are defined in the **VDS_STORAGE_IDENTIFIER_CODE_SET** and **VDS_STORAGE_IDENTIFIER_TYPE** enumerations (section 2.2).

The first structure SHOULD be initialized as follows:

- **m_CodeSet** set to **VDSStorageIdCodeSetBinary**.
- **m_Type** set to **VDSStorageIdTypeFCPHName**.
- **m_rgbIdentifier** set to a **LOGICAL_UNIT_IDENTIFIER** structure (section 2.2.4.9).

The second structure SHOULD be initialized as follows:

- **m_CodeSet** set to **VDSStorageIdCodeSetBinary**.

- **m_Type** set to **VDSStorageIdTypeVendorSpecific**.
- **m_rgbIdentifier** set to a **WINTARGET_DISK_DEVICE_IDENTIFIER** structure (section 2.2.4.21). The **SnapshotId** member of the WINTARGET_DISK_DEVICE_IDENTIFIER structure SHOULD be initialized to zero.

The **VDS_LUN_INFORMATION** structure SHOULD also be initialized with an **m_BusType** member **VDS_STORAGE_BUS_TYPE** value (section 2.2) of **VDSBusTypeScsi**.

3.43.4.39 GetDeviceVolumeGuid (Opnum 41)

The **GetDeviceVolumeGuid** method retrieves the GUID of the volume on which the virtual disk resides.

```
HRESULT GetDeviceVolumeGuid(
        [out] GUID * pGuid);
```

pGuid: A pointer to a **GUID** structure (section 2.2) that returns the GUID.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is configured in an HA cluster, but the resource is offline.
E_INVALIDARG 0x80070057	The LUN is not based on a VHD, or the pointer is invalid.
ERROR_FILE_NOT_FOUND 0x80070002	The LUN references an invalid snapshot

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **GetDeviceVolumeGuid** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- If the virtual disk is not of type **WTDTypeFileBased** (section 2.2.3.10) according to the **Type** data element (section 3.1.1.4), or if the *pGuid* parameter is NULL, the server returns E_INVALIDARG.
- If the virtual disk is a shadow copy, the server finds the volume on which the original resides, returning ERROR_FILE_NOT_FOUND if the original cannot be found.
- Retrieves the GUID of the volume from the **VolumeGuid** data element (section 3.1.1.3.2), and returns it by using the *pGuid* parameter.

3.43.5 Timer Events

None.

3.43.6 Other Local Events

None.

3.44 IWTDisk2 Server Details

The **IWTDisk2** interface extends the **IWTDisk** interface (section 3.43) with support for HA clusters.

3.44.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the LUN management data model (section 3.1.1.4).

3.44.2 Timers

None.

3.44.3 Initialization

LUN manager initialization is specified in section 3.1.3.

3.44.4 Message Processing Events and Sequencing Rules

The **IWTDisk2** interface includes the following methods:

Method	Description
[propget]Enable	Returns the enabled state of the virtual disk. Opnum: 42
[propput]Enable	Sets the enabled state of the virtual disk. Opnum: 43
[propget]ResourceName	Returns the name of the cluster resource associated with the virtual disk. Opnum: 44
[propput]ResourceName	Sets the name of the cluster resource associated with the virtual disk. Opnum: 45
[propget]ResourceGroupName	Returns the name of the cluster resource group associated with the virtual disk.

Method	Description
	Opnum: 46
[propput]ResourceGroupName	Sets the name of the cluster resource group associated with the virtual disk. Opnum: 47
IsAlive	Checks whether the virtual disk is accessible. Opnum: 48
SaveToStream	Saves the virtual disk settings to a specified stream. Opnum: 49

3.44.4.1 [propget]Enable (Opnum 42)

The **[propget]Enable** method returns the enabled state of the virtual disk.

```
[propget] HRESULT Enable(
        [out] [retval] boolean * pEnable);
```

pEnable: A pointer to a Boolean value that returns the enabled state of the virtual disk.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD return the value of the **Enable** data element (section 3.1.1.4) of the virtual disk by using the *pEnable* parameter.

3.44.4.2 [propput]Enable (Opnum 43)

The **[propput]Enable** method sets the enabled state of the virtual disk.

```
[propput] HRESULT Enable(
        [in] boolean bEnable);
```

bEnable: A Boolean value that specifies the enabled state of the virtual disk.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	The underlying device supporting the LUN is incompatible with the operation.<102>
ERROR_SHARING_VIOLATION 0x80070020	The underlying device supporting the LUN is being rolled-back.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]Enable** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- If the enabled state is changing from enabled to disabled, the server SHOULD check whether the VHD is being rolled back, and if it is, the server SHOULD NOT perform the disable operation and SHOULD return ERROR_SHARING_VIOLATION.
- Sets the value of the **Enable** data element (section 3.1.1.4) of the virtual disk from the *bEnable* parameter.
- if the virtual disk is a differencing VHD file, the cache parent flag on the virtual disk SHOULD be set to the current value of the **CacheParent** data element (section 3.1.1.4).
- If the enabled state is changing from enabled to disabled, and the virtual disk is locally mounted, the server SHOULD dismount the virtual disk.
- Sends notifications of the change in the LUN list to registered ISTM clients by using the **IStatusNotify** interface (section 3.42).

3.44.4.3 [propget]ResourceName (Opnum 44)

The **[propget]ResourceName** method returns the name of the cluster resource associated with the virtual disk.

```
[propget] HRESULT ResourceName(
        [out] [retval] BSTR * pResourceName);
```

pResourceName: A pointer to a string that returns the name of the cluster resource associated with the virtual disk. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_INVALIDARG 0x80070057	The supplied pointer is invalid.

When the **[propget]ResourceName** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server returns the value of the **ResourceName** data element (section 3.1.1.4) of the virtual disk by using the *pResourceName* parameter.

3.44.4.4 [propput]ResourceName (Opnum 45)

The **[propput]ResourceName** method sets the name of the cluster resource associated with the virtual disk.

```
[propput] HRESULT ResourceName(
         [in] BSTR ResourceName);
```

ResourceName: The name of the cluster resource associated with the virtual disk.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

Return value/code	Description
E_INVALIDARG 0x80070057	The supplied resource name is invalid.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]ResourceName** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Checks the specified resource name, and if it is not valid SHOULD return E_INVALIDARG.
- Sets the value of the **ResourceName** data element (section 3.1.1.4) of the virtual disk from the *ResourceName* parameter.

3.44.4.5 [propget]ResourceGroupName (Opnum 46)

The **[propget]ResourceGroupName** method returns the name of the cluster resource group associated with the virtual disk.

```
[propget] HRESULT ResourceGroupName(
        [out] [retval] BSTR * pResourceGroup);
```

pResourceGroup: A pointer to a string that returns the name of the cluster resource group associated with the virtual disk. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_POINTER 0x80004003	Invalid pointer.

When the **[propget]ResourceGroupName** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Checks the *pResourceGroupName* parameter; if it is invalid or NULL, the server SHOULD return E_POINTER.
- In an HA cluster configuration, the server returns the **ResourceGroupName** data element (section 3.1.1.4) of the virtual disk by using the *pResourceGroup* parameter.
- In a non-HA cluster configuration, the server SHOULD return an empty string by using the *pResourceGroup* parameter.

3.44.4.6 [propput]ResourceGroupName (Opnum 47)

The **[propput]ResourceGroupName** method MAY set the name of the cluster resource group associated with the virtual disk.<103>

```
[propput] HRESULT ResourceGroupName(
        [in] BSTR ResourceGroup);
```

ResourceGroup: The name of the cluster resource group associated with the virtual disk.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.44.4.7 IsAlive (Opnum 48)

The **IsAlive** method MAY check whether the virtual disk is accessible.<104>

```
HRESULT IsAlive();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed without side effects.

3.44.4.8 SaveToStream (Opnum 49)

The **SaveToStream** method MAY save the virtual disk settings to a specified stream.<105>

```
HRESULT SaveToStream(
        [in] IUnknown * pStream);
```

pStream: A pointer to an **IUnknown** interface DCOM object [MS-DCOM].

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	The operation is not allowed from a remote client.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.

When the **SaveToStream** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION.

The server SHOULD perform any transport and implementation-specific checks that determine the locality of the interface pointer specified by the *pStream* parameter. For a remote caller, the server SHOULD return E_ACCESSDENIED.

Additional processing by this method for a local caller is not required for protocol compliance, and its implementation MAY be omitted.<106>

3.44.5 Timer Events

None.

3.44.6 Other Local Events

None.

3.45 IWTDisk3 Server Details

The **IWTDisk3** interface extends the **IWTDisk2** interface (section 3.44) with additional support for HA clusters and support for differencing VHD files.

3.45.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the LUN management data model (section 3.1.1.4).

3.45.2 Timers

None.

3.45.3 Initialization

LUN manager initialization is specified in section 3.1.3.

3.45.4 Message Processing Events and Sequencing Rules

The **IWTDisk3** interface includes the following methods:

Method	Description
[propget]VhdType	Returns the VHD type of the virtual disk. Opnum: 50
[propget]ParentPath	Returns the parent path of the virtual disk. Opnum: 51
[propget]AllocatedSize	Returns the current allocated size of the VHD. Opnum: 52
[propget]CacheParent	Returns whether the parent VHD was opened with file system caching enabled. Opnum: 53
[propput]CacheParent	Sets whether the parent VHD SHOULD be opened with file system caching enabled. Opnum: 54
[propget]ResourceState	Returns the cluster state of the resource associated with the virtual disk in an HA cluster configuration. Opnum: 55
[propput]ResourceState	Sets the cluster state of the resource associated with the virtual disk in an HA cluster configuration. Opnum: 56
GetLunStatus	Returns the LUN status containing rollback progress information. Opnum: 57

3.45.4.1 [propget]VhdType (Opnum 50)

The **[propget]VhdType** method returns the VHD type of the virtual disk.

```
[propget] HRESULT VhdType(
        [out] [retval] VhdType * Type);
```

Type: A pointer to a **VhdType** value (2.2.3.9) that returns the VHD type of the virtual disk.

Return Values:

The following values are possible:

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.

When the **[propget]VhdType** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD return the value of the **VhdType** data element (section 3.1.1.4) of the virtual disk by using the *Type* parameter.

3.45.4.2 [propget]ParentPath (Opnum 51)

The **[propget]ParentPath** method returns the parent path of the virtual disk. This property is only valid for a differencing VHD.

```
[propget] HRESULT ParentPath(
        [out] [retval] BSTR * ParentPath);
```

ParentPath: A pointer to a string that returns the parent path of the virtual disk.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully, the virtual disk is not based on a VHD, or the VHD is not a differencing VHD.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]ParentPath** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process the method as follows:

- Determines if the property is applicable for the current type of virtual disk and the type of VHD file.
- If the property is not applicable, the server SHOULD return S_OK without assigning any string to the *ParentPath* parameter.
- If the virtual disk type is file-based, and the VHD type is differencing, the server SHOULD determine the parent path of the virtual disk and return it by using the *ParentPath* parameter.

3.45.4.3 [propget]AllocatedSize (Opnum 52)

The **[proget]AllocatedSize** method returns the current allocated size of the VHD.

```
[propget] HRESULT AllocatedSize(
        [out] [retval] hyper * Size);
```

Size: A pointer to a value that returns the currently allocated size of the VHD, in bytes. For a differencing VHD, this value represents the allocated size on the disk.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully, even when the type of virtual disk is not compatible with the operation.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.

When the **[propget]AllocatedSize** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Determines if the property is applicable for the current type of virtual disk and the type of VHD file.
- If the property is not applicable, the server SHOULD return S_OK without assigning any value to the *Size* parameter.
- If the virtual disk type is file-based, and the VHD type is differencing, the server SHOULD return the value of the **AllocatedSize** data element (section 3.1.1.4) of the virtual disk by using the *Size* parameter.

3.45.4.4 [propget]CacheParent (Opnum 53)

The **[propget]CacheParent** method returns whether the parent VHD was opened with file system caching enabled. This property is only applicable to a differencing VHD file.

```
[propget] HRESULT CacheParent(
        [out] [retval] boolean * CacheParent);
```

CacheParent: A pointer to a Boolean value that returns whether the parent VHD was opened with file system caching enabled.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.

When the **[propget]CacheParent** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD return the value of the **CacheParent** data element (section 3.1.1.4) of the virtual disk by using the *CacheParent* parameter.

3.45.4.5 [propput]CacheParent (Opnum 54)

The **[propput]CacheParent** method sets whether the parent VHD SHOULD be opened with file system caching enabled. This property is only applicable to a differencing VHD file.

```
[propput] HRESULT CacheParent(
        [in] boolean CacheParent);
```

CacheParent: A Boolean value that specifies whether the parent VHD SHOULD be opened with file system caching enabled. This property is active the next time the VHD is loaded.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.
E_INVALIDARG 0x80070057	The LUN is not a differencing VHD file.

When the **[propput]CacheParent** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Checks whether the virtual disk is a differencing VHD file, and if it is not, the server SHOULD return E_INVALIDARG.
- Sets the value of the **CacheParent** data element (section 3.1.1.4) of the virtual disk from the *CacheParent* parameter.

3.45.4.6 [propget]ResourceState (Opnum 55)

The **[propget]ResourceState** method returns the cluster state of the resource associated with the virtual disk in an HA cluster configuration.

```
[propget] HRESULT ResourceState(
        [out] [retval] long * ResourceState);
```

ResourceState: A pointer to a value that returns the cluster state of the resource associated with the virtual disk.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Gets the current cluster state of the resource associated with the virtual disk and sets the value of the **ResourceState** data element (section 3.1.1.4) with it.
- Returns the value of the **ResourceState** data element by using the *ResourceState* parameter.

3.45.4.7 [propput]ResourceState (Opnum 56)

The **[propput]ResourceState** method MAY set the cluster state of the resource associated with the virtual disk in an HA cluster configuration. <107>

```
[propput] HRESULT ResourceState(
        [in] long ResourceState);
```

ResourceState: A value that specifies the cluster state of the resource associated with the virtual disk.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.45.4.8 GetLunStatus (Opnum 57)

The **GetLunStatus** method returns the LUN status containing rollback progress information.

```
HRESULT GetLunStatus(
        [out] LUN_STATUS * Status);
```

Status: A pointer to a **LUN_STATUS** structure (section 2.2.4.11) that returns the LUN status containing rollback progress information.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_FAIL 0x80004005	The LUN is disabled or not initialized.
ERROR_INVALID_FUNCTION 0x80070001	The iSCSI Software Target is not configured in an HA cluster, or the resource is offline.

In an HA cluster configuration, the server SHOULD check the **ResourceState** data element (section 3.1.1.4) to determine whether the virtual disk is online, and if the virtual disk is not online, the server SHOULD return ERROR_INVALID_FUNCTION. Otherwise, the server SHOULD process this method as follows:

- Gets the current LUN status of the virtual disk and sets the value of the **LunStatus** data object (section 3.1.1.4) with it.
- Returns the value of the **LunStatus** data object of the virtual disk in a **LUN_STATUS** structure by using the *Status* parameter.

3.45.5 Timer Events

None.

3.45.6 Other Local Events

None.

3.46 IWTDiskMgr Server Details

The **IWTDiskMgr** interface is used to create and manage the LUNs and associated information that have been configured for the iSCSI Software Target.

3.46.1 Abstract Data Model

Unless otherwise specified, in this section the virtual disk term indicates an instance of the **IWTDisk** interface (section 3.43), which is an abstract data element in the **WTDiskList** collection that is maintained by the LUN manager.

For more details concerning the abstract data organization for this interface, see the LUN management data model (section 3.1.1.4).

3.46.2 Timers

None.

3.46.3 Initialization

LUN manager initialization is specified in section 3.1.3.

3.46.4 Message Processing Events and Sequencing Rules

The **IWTDiskMgr** interface includes the following methods:

Method	Description
EnumWTDisks	Returns an enumeration of all virtual disks on the iSCSI Software Target. Opnum: 3
CreateWTDisk	Creates a virtual disk on the iSCSI Software Target. Opnum: 4
CreateShadowWTDisk	Creates a virtual disk on the iSCSI Software Target by using a specified snapshot. Opnum: 5
DeleteWTDisk	Deletes a specified virtual disk on the iSCSI Software Target. Opnum: 6
GetWTDisk	Retrieves a specified virtual disk on the iSCSI Software Target. Opnum: 7
EnumWTDisksOnSameVolumeByPath	Returns an enumeration of all virtual disks on the iSCSI Software Target that reside on the same volume of the specified path. Opnum: 8
EnumWTDisksOnSameVolume	Returns an enumeration of all virtual disks on the iSCSI Software Target that reside on the same volume of the specified virtual disk. Opnum: 9
DVMountSelectedWTDisksSnapshotNow	Mounts all the selected virtual disks on the iSCSI Software Target for a data view snapshot. Opnum: 10
ImportWTDisk	Imports the specified virtual disk into a list of known virtual disks. Opnum: 11

3.46.4.1 EnumWTDisks (Opnum 3)

The **EnumWTDisks** method returns an enumeration of all virtual disks on the iSCSI Software Target.

```
HRESULT EnumWTDisks(
        [out] IEnumWTDisk ** ppEnumWTDisk);
```

ppEnumWTDisk: A pointer to an **IEnumWTDisk** interface (section 3.19) pointer that returns the enumeration of virtual disks.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumWTDisks** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Creates or updates a collection of **IWTDisk** interface objects (section 3.43) that define all virtual disks that exist on the iSCSI Software Target, and saves the result in a **WTDiskList** data object (section 3.1.1.4).
- Creates an **IEnumWTDisk** interface object and initializes it with the collection in **WTDiskList**.
- Returns a pointer to the **IEnumWTDisk** interface by using the *ppEnumWTDisk* parameter.

3.46.4.2 CreateWTDisk (Opnum 4)

The **CreateWTDisk** method creates a virtual disk on the iSCSI Software Target.

```
HRESULT CreateWTDisk(
        [in] WTDType eDeviceType,
        [in] BSTR DevicePath,
        [in] BSTR Description,
        [in] boolean bClearPartTable,
        [in] boolean bForceDismount,
        [out] unsigned long * pWTD);
```

eDeviceType: A **WTDType** value (section 2.2.3.10) that specifies the type of virtual disk to create.<108>

DevicePath: The full path to the virtual disk device.

Description: A user-friendly description for the virtual disk.

bClearPartTable: A Boolean value that specifies whether to set to zero the **partition table** of the physical disk.

bForceDismount: A Boolean value that specifies whether the volume is to be forcibly dismounted. This parameter SHOULD be ignored.

pWTD: A pointer to a value that returns the disk index of the virtual disk that is created. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_POINTER 0x80004003	Invalid pointer.
E_FAIL 0x80004005	The specified device path is malformed or invalid.
ERROR_DEVICE_IN_USE 0x80070964	The specified device path is in use by a different LUN.
ERROR_ALREADY_EXISTS 0x800700b7	The specified device path is already present.
ERROR_BAD_LENGTH 0x80070018	The size of the file specified is incompatible with implementation limits.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) defined in section 2.2.1 or [MS-ERREF].

When the **CreateWTDisk** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Validates the pointers in the parameters *DevicePath, Description*, and *pWTD,* and returns E_POINTER if any of them are NULL.
- Verifies that if the device type specified in *eDeviceType* is **WTDTypeFileBased** (section 2.2.3.10), the device path specified in *DevicePath* is a path to a file.
- Checks whether the device path is already being used for a virtual disk, and if it is, returns ERROR_ALREADY_EXISTS.
- Performs additional checks on the *DevicePath* parameter to ensure it can be used for the new virtual disk and returns an appropriate error if it cannot.
- In an HA cluster environment, performs the following additional processing:
- If the VHD of the virtual disk is a **DifferencingVhd** (section 2.2.3.9), the server SHOULD ensure that the physical disk resources of the VHD and the parent VHD of the virtual disk are in the same resource group.

- Creates a resource for the new virtual disk in the cluster.
- Creates a dependency from the virtual disk resource to the physical disk resource to ensure that the physical disk is online before the virtual disk. If the VHD of the virtual disk is a **DifferencingVhd**, a similar dependency SHOULD be created for the parent physical disk.
- Brings the new virtual disk online.
- Clears the partition table of the disk with the specified *DevicePath* if indicated by the *bClearPartTable* parameter.
- Creates an instance of the **IWTDisk** interface (section 3.43) and initializes the properties of the virtual disk described in section 3.1.1.4.
- Adds the IWTDisk object to the collection of virtual disks in the **WTDiskList** data object (section 3.1.1.4).
- Sends notifications of the change in the LUN list to registered ISTM clients by using the **IStatusNotify** interface (section 3.42).
- Saves the settings for the new virtual disk to persistent storage, so that they will continue to be defined after a system restart.
- Sets the snapshot storage location on the VHD represented by the device path.
- Returns the disk index of the new virtual disk by using the *pWTD* parameter.

3.46.4.3 CreateShadowWTDisk (Opnum 5)

The **CreateShadowWTDisk** method MAY create a virtual disk on the iSCSI Software Target by using a specified snapshot.<109>

```
HRESULT CreateShadowWTDisk(
        [in] WTDType eDeviceType,
        [in] BSTR DevicePath,
        [in] BSTR Description,
        [in] GUID SnapshotId,
        [out] unsigned long * pWTD);
```

eDeviceType: A **WTDType** value (section 2.2.3.10) that specifies the type of virtual disk to create.<110>

DevicePath: The full path to the virtual disk device.

Description: A user-friendly description for the virtual disk.

SnapshotId: A **GUID** structure (section 2.2) that identifies a snapshot.

pWTD: A pointer to a value that returns the disk index of the virtual disk that is created.

Return Values:

The following values are possible:

Return value/code	Description
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_NOTIMPL 0x80004001	Not implemented.

When the **CreateShadowWTDisk** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary. If the preconditions checks are successful, the server SHOULD return E_NOTIMPL.

3.46.4.4 DeleteWTDisk (Opnum 6)

The **DeleteWTDisk** method deletes a specified virtual disk on the iSCSI Software Target.

```
HRESULT DeleteWTDisk(
        [in] unsigned long nWTD);
```

nWTD: The disk index of the virtual disk to delete.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_DRIVE_LOCKED 0x8007006c	Delete disallowed. There could be initiators connected to the target using the LUN.
ERROR_SHARING_VIOLATION 0x80070020	The present LUN is locally mounted.
ERROR_DEPENDENT_RESOURCE_EXISTS 0x80071389	The snapshot related to the LUN is locally mounted.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **DeleteWTDisk** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Checks whether the disk index specified in the *nWTD* parameter is valid in the **IWTDiskList** collection. If it is not valid, the server SHOULD return S_OK.
- Checks whether the virtual disk is in use by any iSCSI initiator. If the virtual disk is in use, the server SHOULD return ERROR_DRIVE_LOCKED.
- Checks whether the virtual disk is in use by a snapshot that is locally mounted. If the virtual disk is in use, the server SHOULD return ERROR_SHARING_VIOLATION.
- Checks whether the virtual disk is in use by a snapshot that is exported as a virtual disk. If the virtual disk is in use, the server SHOULD return ERROR_DEPENDENT_RESOURCE_EXISTS.
- In an HA cluster environment, obtains the virtual disk cluster resource and deletes it, after also deleting any dependencies on the physical disk resource.
- Unmaps the virtual disk from all targets.
- Notifies the snapshot manager that the virtual disk has been removed.
- Removes the **IWTDisk** object associated with the virtual disk from the collection maintained in the **WTDiskList** data object (section 3.1.1.4).
- Sends notifications of the change in the LUN list to registered ISTM clients by using the **IStatusNotify** interface (section 3.42).

3.46.4.5 GetWTDisk (Opnum 7)

The **GetWTDisk** method retrieves a specified virtual disk on the iSCSI Software Target.

```
HRESULT GetWTDisk(
        [in] unsigned long nWTD,
        [out] IWTDisk ** ppWTDisk);
```

nWTD: The disk index of the virtual disk to retrieve.

ppWTDisk: A pointer to an **IWTDisk** interface (section 3.43) pointer that returns the specified virtual disk.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

Return value/code	Description
ERROR_HOST_NODE_NOT_RESOURCE_OWNER 0x80071397	In an HA cluster configuration, the LUN resource is not owned by the node.
ERROR_FILE_NOT_FOUND 0x80070002	The LUN identified by the specified index could not be found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **GetWTDisk** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Uses the specified disk index to retrieve the instance of the IWTDisk interface for the virtual disk from the collection maintained in the **WTDiskList** data object (section 3.1.1.4).
- In an HA cluster configuration, if the cluster resource for the specified virtual disk is offline, the virtual disk will not be found in the **WTDiskList** collection. If this is the case, the server SHOULD perform the following processing actions:
- Checks whether the cluster resource exists. If it does not exist, the server SHOULD return ERROR_FILE_NOT_FOUND.
- Check whether the cluster resource for the specified virtual disk is owned by this iSCSI Software Target computer. If it is not, the server SHOULD return ERROR_HOST_NODE_NOT_RESOURCE_OWNER.
- Otherwise, the server SHOULD create a new instance of the **IWTDisk** interface to return from this method without adding it to the **WTDiskList** collection. The **IWTDisk** element returned SHOULD only have the **WTDIndex**, **ResorurceName** and **ResoruceState** abstract data model fields initialized (section 3.1.1.4).
- In a non- HA cluster configuration, if the specified virtual disk is not found in the **WTDiskList** collection, the server SHOULD return ERROR_FILE_NOT_FOUND.
- Returns a pointer to the **IWTDisk** interface for the virtual disk by using the *ppWTDisk* parameter.

3.46.4.6 EnumWTDisksOnSameVolumeByPath (Opnum 8)

The **EnumWTDisksOnSameVolumeByPath** method MAY return an enumeration of all virtual disks on the iSCSI Software Target that reside on the same volume by path.<111>

```
HRESULT EnumWTDisksOnSameVolumeByPath(
        [in] BSTR Path,
        [out] IEnumWTDisk ** ppEnumWTDisk);
```

Path: The path.

ppEnumWTDisk: A pointer to an **IEnumWTDisk** interface (section 3.19) pointer that returns the enumeration of virtual disks.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.46.4.7 EnumWTDisksOnSameVolume (Opnum 9)

The **EnumWTDisksOnSameVolume** method returns an enumeration of all virtual disks on the iSCSI Software Target that reside on the same volume as the specified virtual disk.

```
HRESULT EnumWTDisksOnSameVolume(
        [in] unsigned long nWTD,
        [out] IEnumWTDisk ** ppEnumWTDisk);
```

nWTD: The disk index of the virtual disk.

ppEnumWTDisk: A pointer to an **IEnumWTDisk** interface (section 3.19) pointer that returns the enumeration of virtual disks.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_FILE_NOT_FOUND 0x80070002	The LUN identified by the specified index could not be found.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumWTDisksOnSameVolume** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Obtains the device path property for the virtual disk specified by the *nWTD* parameter.

- Creates a collection of **IWTDisk** interface objects (section 3.19) that includes all virtual disks on the iSCSI Software Target that reside on the volume specified by the device path property just obtained, and saves the result in a **WTDiskList** data object (section 3.1.1.4).
- Creates an **IEnumWTDisk** interface object and initializes it with the collection in **WTDiskList**.
- Returns a pointer to the **IEnumWTDisk** interface by using the *ppEnumWTDisk* parameter.

3.46.4.8 DVMountSelectedWTDisksSnapshotNow (Opnum 10)

The **DVMountSelectedWTDisksSnapshotNow** method MAY mount all the selected virtual disks on the iSCSI Software Target for a data view snapshot.<112>

```
HRESULT DVMountSelectedWTDisksSnapshotNow();
```

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed but it had no side effects.

3.46.4.9 ImportWTDisk (Opnum 11)

The **ImportWTDisk** method MAY import the specified virtual disk into list of known virtual disks.<113>

```
HRESULT ImportWTDisk(
        [in] IWTDisk * pWTDisk);
```

pWTDisk: A pointer to an **IWTDisk** interface (section 3.43) object that specifies the virtual disk to import.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.46.5 Timer Events

None.

3.46.6 Other Local Events

None.

3.47 IWTDiskMgr Client Details

The **IWTDiskMgr** interface (section 3.46) is used to create and manage the LUNs and associated information that have been configured for the iSCSI Software Target.

3.47.1 Abstract Data Model

The client data model is specified in section 3.2.1.

3.47.2 Timers

None.

3.47.3 Initialization

Client initialization is specified in section 3.2.3.

3.47.4 Message Processing Events and Sequencing Rules

This section specifies client processing of error returns by methods of the **IWTDiskMgr** interface (section 3.46).

3.47.4.1 CreateWTDisk (Opnum 4)

Processing of the following error returns SHOULD be performed by client callers of this server method:

ERROR_ALREADY_EXISTS

A return value of ERROR_ALREADY_EXISTS indicates that an attempt has been made to create a virtual disk with the name of a virtual disk that already exists.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the specified virtual disk name cannot be used.<114>

ERROR_BAD_LENGTH

A return value of ERROR_BAD_LENGTH indicates that an attempt has been made to create a virtual disk with a size that is not within implementation-defined limits.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that a virtual disk of the specified size cannot be created.<115>

3.47.4.2 DeleteWTDisk (Opnum 6)

Processing of the following error returns SHOULD be performed by client callers of this server method:

ERROR_DRIVE_LOCKED

A return value of ERROR_DRIVE_LOCKED indicates that an attempt has been made to delete a virtual disk that is currently in use by another process. Some of the conditions that cause this error are transient, and the operation can be retried at a later time.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the specified virtual disk cannot be deleted, and it SHOULD identify a possible remedy for the failure.<116>

ERROR_SHARING_VIOLATION

A return value of ERROR_SHARING_VIOLATION indicates that the virtual disk is currently in use by another process. Some of the conditions that cause this error are transient, and the operation can be retried at a later time.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the virtual disk deletion failed, and it SHOULD identify a possible remedy for the failure.<117>

ERROR_DEPENDENT_RESOURCE_EXISTS

A return value of ERROR_DEPENDENT_RESOURCE_EXISTS indicates that an attempt has been made to delete a virtual disk that has a dependency that cannot be removed.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the specified virtual disk cannot be deleted, and it SHOULD identify a possible cause of the failure.<118>

3.47.5 Timer Events

None.

3.47.6 Other Local Events

None.

3.48 IWTDiskMgr2 Server Details

The **IWTDiskMgr2** interface extends the **IWTDiskMgr** interface (section 3.46) with support for non-persistent virtual disk configurations.

3.48.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the LUN management abstract data model (section 3.1.1.4).

3.48.2 Timers

None.

3.48.3 Initialization

LUN manager initialization is specified in section 3.1.3.

3.48.4 Message Processing Events and Sequencing Rules

The **IWTDiskMgr2** interface includes the following methods:

Method	Description
CreateWTDiskNonPersistent	Creates a virtual disk but does not save it to persistent storage.

Method	Description
	Opnum: 12
CreateShadowWTDiskNonPersistent	Creates a virtual disk by using a specified snapshot but does not save it to persistent storage. Opnum: 13
OpenWTDisk	Creates a virtual disk by loading its settings from a stream object. Opnum: 14
CloseWTDisk	Closes a specified virtual disk. Opnum: 15

3.48.4.1 CreateWTDiskNonPersistent (Opnum 12)

The **CreateWTDiskNonPersistent** method creates a virtual disk but does not save it to persistent storage. That is, after a system restart, the virtual disk created by this method will no longer be defined.

```
HRESULT CreateWTDiskNonPersistent(
        [in] unsigned long nWTD,
        [in] WTDType eDeviceType,
        [in] BSTR DevicePath,
        [in] BSTR Description,
        [in] boolean bClearPartTable,
        [in] boolean bForceDismount,
        [in] BSTR ResourceName,
        [out] IWTDisk ** ppWTDisk);
```

nWTD: The disk index to be used for the new virtual disk.

eDeviceType: A **WTDType** value (section 2.2.3.10) that specifies the type of virtual disk to create.<119>

DevicePath: The full path to the virtual disk device.

Description: A user-friendly description for the virtual disk.

bClearPartTable: A Boolean value that specifies whether to set to zero the partition table of the physical disk.

bForceDismount: A Boolean value that specifies whether the volume is to be forcibly dismounted. This parameter SHOULD be ignored.

ResourceName: The name of the cluster resource that represents this virtual disk.

ppWTDisk: A pointer to an **IWTDisk** interface (section 3.43) pointer that returns the virtual disk that is created. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
E_INVALIDARG 0x80070057	The device path is malformed, or it points to a file format that is incompatible with the operation.
E_POINTER 0x80004003	Invalid pointer.
ERROR_ALREADY_EXISTS 0x800700b7	The specified device path is already in use by a LUN.
ERROR_BAD_LENGTH 0x80070018	The specified device path designates a file that is an invalid size.
ERROR_FILE_NOT_FOUND 0x80070002	The system cannot find the file specified.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

The server SHOULD process this method as follows:

- Validates the pointers in the parameters *DevicePath, Description*, and *ppWTDisk*, and returns E_POINTER if any of the parameters are NULL.
- Verifies that if the device type specified in *eDeviceType* is **WTDTypeFileBased** (section 2.2.3.10), the device path specified in *DevicePath* is a path to a file.
- Checks whether the device path is already being used for a virtual disk, and if it is, returns ERROR_ALREADY_EXISTS.
- Performs additional checks on the *DevicePath* parameter to ensure it can be used for the new virtual disk and returns an appropriate error if it cannot.
- Checks the disk index specified in *nWTD*. If it is already being used by a virtual disk, the server SHOULD return ERROR_ALREADY_EXISTS. If the value of *nWTD* is -1, the server SHOULD use the next available index for the new virtual disk.
- Clears the partition table of the disk with the specified *DevicePath* if indicated by the *bClearPartTable* parameter.
- Creates an instance of the **IWTDisk** interface (section 3.43) and initializes the properties of the virtual disk defined in section 3.1.1.4.
- Sets the value of the **ResourceState** data element (section 3.1.1.4) to **ClusterResourceOnline** ([MS-CMRP] section 3.1.4.1.13).
- Adds the **IWTDisk** object to the collection of virtual disks in the **WTDiskList** data object (section 3.1.1.4).
- Sends notifications of the change in the LUN list to registered ISTM clients by using the **IStatusNotify** interface (section 3.42).

- Returns the pointer to the **IWTDisk** interface for the new virtual disk by using the *ppWTDisk* parameter.

3.48.4.2 CreateShadowWTDiskNonPersistent (Opnum 13)

The **CreateShadowWTDiskNonPersistent** method creates a virtual disk by using a specified snapshot but does not save it to persistent storage. That is, after a system restart, the virtual disk created by this method will no longer be defined.

```
HRESULT CreateShadowWTDiskNonPersistent(
        [in] unsigned long nWTD,
        [in] WTDType eDeviceType,
        [in] BSTR DevicePath,
        [in] BSTR Description,
        [in] GUID SnapshotId,
        [in] BSTR ResourceName,
        [out] IWTDisk ** ppWTDisk);
```

nWTD: The disk index to be used for the new virtual disk.

eDeviceType: A **WTDType** value (section 2.2.3.10) that specifies the type of virtual disk to create.<120>

DevicePath: The full path to the virtual disk device.

Description: A user-friendly description for the virtual disk.

SnapshotId: A **GUID** structure (section 2.2) that specifies the snapshot to use in creating the virtual disk. A value of zero indicates that the disk is not backed up by a volume snapshot.

ResourceName: The name of the resource group that owns the new virtual disk.

ppWTDisk: A pointer to an **IWTDisk** interface (section 3.43) pointer that returns the virtual disk that is created. This parameter MUST NOT be NULL.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
E_POINTER 0x80004003	Invalid pointer.
ERROR_ALREADY_EXISTS	The specified DevicePath is already in use by a LUN, or the Disk index is already in use.

Return value/code	Description
0x800700b7	
ERROR_FILE_NOT_FOUND 0x80070002	The system cannot find the file specified.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **CreateShadowWTDiskNonPersistent** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Validates the pointers in the parameters *DevicePath, Description*, and *ppWTDisk*, and returns E_POINTER if any of the parameters are NULL.
- Verifies that if the device type specified in *eDeviceType* is **WTDTypeFileBased** (section 2.2.3.10), the device path specified in *DevicePath* is a path to a file.
- Checks whether the device path is already being used for a virtual disk, and if it is, returns ERROR_ALREADY_EXISTS.
- Performs additional checks on the *DevicePath* parameter to ensure it can be used for the new virtual disk and returns an appropriate error if it cannot.
- Checks the disk index specified in *nWTD*. If it is already being used by a virtual disk, the server SHOULD return ERROR_ALREADY_EXISTS. If the value of *nWTD* is -1, the server SHOULD use the next available index for the new virtual disk.
- Creates an instance of the **IWTDisk** interface (section 3.43) with the snapshot specified by *SnapshotId* representing the physical disk.
- Initializes the properties of the virtual disk defined in section 3.1.1.4.
- Sets the value of the **ResourceState** data element (section 3.1.1.4) to **ClusterResourceOnline** ([MS-CMRP] section 3.1.4.1.13).
- Sends notifications of the change in the LUN list to registered ISTM clients by using the **IStatusNotify** interface (section 3.42).
- Returns the pointer to the **IWTDisk** interface for the new virtual disk by using the *ppWTDisk* parameter.

3.48.4.3 OpenWTDisk (Opnum 14)

The **OpenWTDisk** method MAY create a virtual disk by loading its settings from a stream object.<121>

```
HRESULT OpenWTDisk(
        [in] IUnknown * pStream,
        [in] BSTR ResourceName,
        [out] IWTDisk ** ppWTDisk);
```

pStream: A pointer to an **IUnknown** interface DCOM object [MS-DCOM] that contains the virtual disk settings. The data contained or referenced as a payload in the stream is undefined, since this method does not process the data.

ResourceName: The name of the cluster resource that represents the virtual disk.

ppWTDisk: A pointer to an **IWTDisk** interface (section 3.43) pointer.

Return Values:

The following values are possible:

Return value/code	Description
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	The client is not authorized to perform this call or the client is a remote client.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **OpenWTDisk** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD perform any transport and implementation-specific checks that determines the locality of the interface pointer specified by the *pStream* parameter. For a remote caller, the server SHOULD return E_ACCESSDENIED.

Additional processing by this method for a local caller is not required for protocol compliance, and its implementation MAY be omitted.<122>

3.48.4.4 CloseWTDisk (Opnum 15)

The **CloseWTDisk** method closes a specified virtual disk.

```
HRESULT CloseWTDisk(
        [in] unsigned long nWTD);
```

nWTD: The disk index of the virtual disk to close.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation succeeded, or the specified disk was not found.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED	General access denied error.

Return value/code	Description
0x80070005	

When the **CloseWTDisk** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- Finds the **IWTDisk** object (section 3.43) with the disk index specified by the *nWTD* parameter in the collection of virtual disks in the **WTDiskList** data object (section 3.1.1.4).
- Removes the virtual disk from the **WTDiskList** collection.
- Dismounts the virtual disks from data view.
- Aborts any rollback in progress.
- Unmaps the virtual disk from all iSCSI targets.
- Closes any snapshots associated with the virtual disk,
- Sends notifications of the change in the LUN list to registered ISTM clients by using the **IStatusNotify** interface (section 3.42).

3.48.5 Timer Events

None.

3.48.6 Other Local Events

None.

3.49 IWTDiskMgr2 Client Details

The **IWTDiskMgr2** interface (section 3.48) extends the **IWTDiskMgr** interface (section 3.46) with support for snapshots.

3.49.1 Abstract Data Model

The client data model is specified in section 3.2.1.

3.49.2 Timers

None.

3.49.3 Initialization

Client initialization is specified in section 3.2.3.

3.49.4 Message Processing Events and Sequencing Rules

This section specifies client processing of error returns by methods of the **IWTDiskMgr2** interface (section 3.48).

3.49.4.1 CreateWTDiskNonPersistent (Opnum 12)

Processing of the following error returns SHOULD be performed by client callers of this server method:

ERROR_ALREADY_EXISTS

A return value of ERROR_ALREADY_EXISTS indicates that an attempt has been made to create a virtual disk with a device path or disk index that is already in use by a LUN.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the specified virtual disk cannot be created.

ERROR_BAD_LENGTH

A return value of ERROR_BAD_LENGTH indicates that an attempt has been made to create a virtual disk with a device path that designates a file of a size that is not within implementation-defined limits.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that a virtual disk for a file of the specified size cannot be created.

ERROR_FILE_NOT_FOUND

A return value of ERROR_FILE_NOT_FOUND indicates that an attempt has been made to create a virtual disk with a device path that designates a file that cannot be found.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that a virtual disk with the specified device path cannot be created.

3.49.4.2 CreateShadowWTDiskNonPersistent (Opnum 13)

Processing of the following error returns SHOULD be performed by client callers of this server method:

ERROR_ALREADY_EXISTS

A return value of ERROR_ALREADY_EXISTS indicates that an attempt has been made to create a virtual disk with a device path or disk index that is already in use by a LUN.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the specified virtual disk cannot be created.

ERROR_FILE_NOT_FOUND

A return value of ERROR_FILE_NOT_FOUND indicates that an attempt has been made to create a virtual disk with a device path that designates a file that cannot be found.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that a virtual disk with the specified device path cannot be created.

3.49.5 Timer Events

None.

3.49.6 Other Local Events

None.

3.50 IWTDiskMgr3 Server Details

The **IWTDiskMgr3** interface extends the **IWTDiskMgr2** interface (section 3.48) with support for HA clusters.

3.50.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the LUN management data model (section 3.1.1.4).

3.50.2 Timers

None.

3.50.3 Initialization

LUN manager initialization is specified in section 3.1.3.

3.50.4 Message Processing Events and Sequencing Rules

The **IWTDiskMgr3** interface includes the following method:

Method	Description
ClusterNotifyWTDiskDeletion	Performs any ancillary activity that is triggered by a virtual disk deletion in an HA cluster configuration. Opnum: 16

3.50.4.1 ClusterNotifyWTDiskDeletion (Opnum 16)

The **ClusterNotifyWTDiskDeletion** method MAY perform any ancillary activity that is triggered by a virtual disk deletion in an HA cluster configuration.<123>

```
HRESULT ClusterNotifyWTDiskDeletion(
        [in] unsigned long nWTD);
```

nWTD: The disk index of the virtual disk to be deleted.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.50.5 Timer Events

None.

3.50.6 Other Local Events

None.

3.51 IWTDiskMgr4 Server Details

The **IWTDiskMgr4** interface extends the **IWTDiskMgr3** interface (section 3.50) with additional support for HA clusters.

3.51.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the LUN management data model (section 3.1.1.4).

3.51.2 Timers

None.

3.51.3 Initialization

LUN manager initialization is specified in section 3.1.3.

3.51.4 Message Processing Events and Sequencing Rules

The **IWTDiskMgr4** interface includes the following methods:

Method	Description
EnumAllClusterWTDisks	Returns an enumeration of all virtual disks in an HA cluster configuration, including those that are offline. Opnum: 17
ClusterNotifyWTDiskDeletion2	Performs any ancillary activity that is triggered by a virtual disk deletion in an HA cluster configuration. Opnum: 18

3.51.4.1 EnumAllClusterWTDisks (Opnum 17)

The **EnumAllClusterWTDisks** method returns an enumeration of all virtual disks in an HA cluster configuration, including those that are offline.

```
HRESULT EnumAllClusterWTDisks(
        [out] IEnumWTDisk ** EnumWTDisk);
```

EnumWTDisk: A pointer to an **IEnumWTDisk** interface (section 3.19) pointer that returns the enumeration of virtual disks.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumAllClusterWTDisks** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

If not in an HA cluster configuration, this function is the same as the **EnumWTDisks** method (section 3.46.4.1). Otherwise, the server SHOULD process this method as follows:

- Creates **IWTDisk** interface objects (section 3.19) for all virtual disks that are offline and are in the HA cluster containing the node running the iSCSI Software Target.
- Adds the **IWTDisk** interface objects to the collection of online virtual disks in the **WTDiskList** data object (section 3.1.1.4), if one exists. If a **WTDiskList** collection does not already exist, one SHOULD be created.
- Creates an **IEnumWTDisk** interface object and initializes it with the collection in **WTDiskList**.
- Returns a pointer to the **IEnumWTDisk** interface by using the *EnumWTDisk* parameter.

3.51.4.2 ClusterNotifyWTDiskDeletion2 (Opnum 18)

The **ClusterNotifyWTDiskDeletion** method performs any ancillary activity that is triggered by a virtual disk deletion in an HA cluster configuration.

```
HRESULT ClusterNotifyWTDiskDeletion2(
        [in] unsigned long nWTD,
        [in] boolean bShadowDisk);
```

nWTD: The disk index of the virtual disk to be deleted.

bShadowDisk: A Boolean value that specifies whether the virtual disk referenced by the *nWTD* parameter was exported from a snapshot.

Return Values:

The following values are possible.

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.
ERROR_INVALID_OPERATION 0x800710dd	The method was called in a non HA-Configuration.
ERROR_INVALID_PARAMETER 0x80070057	The LUN identified by the specified index could be found.

For any other condition, this method SHOULD return a **HRESULT** (section 2.2) value [MS-ERREF].

When the **EnumAllClusterWTDisks** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD process this method as follows:

- If the *bShadowDisk* parameter is set to true, enumerates all snapshots on the iSCSI Software Target in order to find any that were exported with the disk index specified by the *nWTD* parameter.
- If any corresponding snapshot is found, its associated virtual disk SHOULD be deleted.
- Updates the collection of virtual disks in the **WTDiskList** data object (section 3.1.1.4) to reflect the virtual disk deletion.

3.51.5 Timer Events

None.

3.51.6 Other Local Events

None.

3.52 IWTGeneral Server Details

The **IWTGeneral** interface is used to maintain global settings and statuses.

3.52.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the feature management data model (section 3.1.1.1).

3.52.2 Timers

None.

3.52.3 Initialization

Feature manager initialization is specified in section 3.1.3.

3.52.4 Message Processing Events and Sequencing Rules

The **IWTGeneral** interface includes the following methods:

Method	Description
[propget]Guid	Returns a GUID that identifies the ISTM server service. Opnum: 3
[propget]Name	Returns a name that identifies the iSCSI Software Target. Opnum: 4
[propput]Name	Sets a name that identifies the iSCSI Software Target. Opnum: 5
[propget]License	Returns licensing information for the iSCSI Software Target. Opnum: 6
[propget]WTCliPath	Returns the path name of an application that can schedule a snapshot backup. Opnum: 7
CheckAccess	Checks preconditions that are required for the processing of an ISTM server method call. Opnum: 8
AddLicense	Configures licensing information for the iSCSI Software Target. Opnum: 9
IsSnapshotsSupported	Returns whether snapshot backup functionality is supported. Opnum: 10
RegisterNotification	Registers a notification interface for status updates. Opnum: 11
UnregisterNotification	Removes a prior registration for status updates. Opnum: 12
DVGetSnapshotSchedule	Retrieves the current snapshot schedule. Opnum: 13
DVSetSnapshotSchedule	Sets the current snapshot schedule. Opnum: 14

3.52.4.1 [propget]Guid (Opnum 3)

The **[propget]Guid** method returns a GUID that identifies the ISTM server service.

```
[propget] HRESULT Guid(
          [out] [retval] GUID * pGuid);
```

pGuid: A pointer to a **GUID** structure (section 2.2) that returns the identifier. This value MUST be unique to the instance of ISTM server service that is running.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Guid** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD obtain the GUID from the **Guid** data object (section 3.1.1.1) and return it by using the *pGuid* parameter.

3.52.4.2 [propget]Name (Opnum 4)

The **[propget]Name** method returns a string that identifies the iSCSI Software Target.

```
[propget] HRESULT Name(
        [out] [retval] BSTR * pName);
```

pName: A pointer to a string that returns a descriptive name of the iSCSI Software Target. This value SHOULD uniquely identify this iSCSI Software Target on the operating system.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propget]Name** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The server SHOULD obtain the string from the **Name** data object (section 3.1.1.1) and return it by using the *pName* parameter. This name can be specified by the **[propput]Name** method (section 3.52.4.3). If the name has not been specified, a default value MUST be returned.<124>

3.52.4.3 [propput]Name (Opnum 5)

The **[propput]Name** method sets a name that identifies the iSCSI Software Target.

```
[propput] HRESULT Name(
        [in] BSTR Name);
```

Name: A name that identifies the iSCSI Software Target. This domain of uniqueness of this value SHOULD be the operating system.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **[propput]Name** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The specified string is saved in the **Name** data object (section 3.1.1.1).

3.52.4.4 [propget]License (Opnum 6)

The **[propget]License** method MAY return licensing information for the iSCSI Software Target.<125>

```
[propget] HRESULT License(
        [out] [retval] LICENSE * pLicense);
```

pLicense: A pointer to the **LICENSE** structure (section 2.2.4.8) that returns the licensing information.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.52.4.5 [propget]WTCliPath (Opnum 7)

The **[propget]WTCliPath** method SHOULD return the path name of an application that can schedule a snapshot backup.<126>

```
[propget] HRESULT WTCliPath(
        [out] BSTR * pWtCliPath);
```

pWtCliPath: A pointer to a string that returns the operating system path name of an executable file.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

3.52.4.6 CheckAccess (Opnum 8)

The **CheckAccess** method checks preconditions that are required for the processing of an ISTM server method call.

```
HRESULT CheckAccess();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	ISTM cluster services are currently unavailable.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

See **Check Preconditions** (section 3.1.4.2.1) for a specification of the processing performed by this method.

3.52.4.7 AddLicense (Opnum 9)

The **AddLicense** method MAY configure licensing information for the iSCSI Software Target.<127>

```
HRESULT AddLicense(
        [in] BSTR License);
```

License: A string that contains licensing information.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.52.4.8 IsSnapshotsSupported (Opnum 10)

The **IsSnapshotsSupported** method returns whether snapshot backup functionality is supported.

```
HRESULT IsSnapshotsSupported();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The backup functionality is supported.
S_FALSE 0x00000001	The backup functionality is not supported.

Snapshot functionality requires an infrastructure capable of creating snapshots of LUNs, similar to the one provided by a volume shadow copy service (VSS). The functionality might not be available on all operating systems running the iSCSI Software Target.

3.52.4.9 RegisterNotification (Opnum 11)

The **RegisterNotification** method registers a notification interface for status updates.

```
HRESULT RegisterNotification(
        [in] IStatusNotify * pStatusNotify,
         [in] unsigned long lCookie);
```

pStatusNotify: A pointer to an **IStatusNotify** interface (section 3.41), which defines notification methods to call for status updates.

lCookie: An identifier that is unique to this registration, which is passed to the ISTM client in notification method invocations.

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.
ERROR_NOT_READY 0x80070015	The cluster service might not be running or is temporarily unavailable, or a transient condition makes the service unavailable.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **RegisterNotification** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

The cookie that is associated with the notification interface allows an ISTM client to have multiple callback registrations for the same instance of the **IStatusNotify** interface; that is, the same interface can be registered multiple times with different values of the *lCookie* parameter.

3.52.4.10 UnregisterNotification (Opnum 12)

The **UnregisterNotification** method removes a prior registration for status updates. All registrations that match the specified status notification interface pointer are removed.

```
HRESULT UnregisterNotification(
         [in] IStatusNotify * pStatusNotify);
```

pStatusNotify: A pointer to an **IStatusNotify** interface (section 3.41), which was previously registered for status updates.

Return Values:

The following values are possible:

Return value/code	Description
S_OK	The operation completed successfully.

Return value/code	Description
0x00000000	
ERROR_NOT_READY 0x80070015	The device is not ready.
E_ACCESSDENIED 0x80070005	General access denied error.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

When the **UnregisterNotification** method is called, the ISTM server SHOULD perform the preconditions checks that are specified in section 3.1.4.2.1, returning an appropriate error as necessary.

If multiple prior calls to **RegisterNotification** (section 3.52.4.9) were made on the specified **IStatusNotify** interface with various cookie parameters, all such registrations MUST be removed.

3.52.4.11 DVGetSnapshotSchedule (Opnum 13)

The **DVGetSnapshotSchedule** method MAY return the current snapshot schedule.<128>

```
HRESULT DVGetSnapshotSchedule(
        [in, out] SNAPSHOT_SCHEDULE_SETTINGS * pSchedule);
```

pSchedule: A pointer to a **SNAPSHOT_SCHEDULE_SETTINGS** structure (section 2.2.4.17) that returns the current snapshot schedule.

Return Values:

The following value is possible:

Return value/code	Description
E_NOTIMPL 0x80004001	Not implemented.

3.52.4.12 DVSetSnapshotSchedule (Opnum 14)

The **DVSetSnapshotSchedule** method sets the current snapshot schedule.

```
HRESULT DVSetSnapshotSchedule(
        [in] SNAPSHOT_SCHEDULE_SETTINGS Schedule);
```

Schedule: A **SNAPSHOT_SCHEDULE_SETTINGS** structure (section 2.2.4.17) that defines the current snapshot schedule.

Return Values:

The following value is possible:

Return value/code	Description
S_OK 0x00000000	The operation completed successfully.

For any other condition, this method SHOULD return an **HRESULT** (section 2.2) value [MS-ERREF].

Only one snapshot schedule at a time is active. If the **hyStartTime** member of the **SNAPSHOT_SCHEDULE_SETTINGS** structure is zero, a snapshot is not scheduled.

The specified snapshot schedule is saved in a **Schedule** data object, which is added to the **ScheduleList** data object (section 3.1.1.1).

The name of the application that performs the snapshot can be obtained using the **[propget]WTCliPath** method (section 3.52.4.5).

3.52.5 Timer Events

None.

3.52.6 Other Local Events

None.

3.53 IWTGeneral Client Details

The **IWTGeneral** interface (section 3.52) is used to maintain global settings and statuses.

3.53.1 Abstract Data Model

The client data model is specified in section 3.2.1.

3.53.2 Timers

None.

3.53.3 Initialization

Client initialization is specified in section 3.2.3.

3.53.4 Message Processing Events and Sequencing Rules

This section specifies client processing of error returns by methods of the **IWTGeneral** interface (section 3.52).

3.53.4.1 CheckAccess (Opnum 8)

Processing of the following error returns SHOULD be performed by client callers of this server method:

ERROR_NOT_READY

A return value of ERROR_NOT_READY indicates that the required iSCSI Software Target services are not available. See **Check Preconditions** (section 3.1.4.2.1) for details of the conditions that can result in this return. Some of the conditions that cause this error are transient, and the operation can be retried at a later time, after a new interface pointer has been obtained through DCOM instantiation.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that iSCSI Software Target services are not available.<129>

E_ACCESSDENIED

A return value of E_ACCESSDENIED indicates that the ISTM client is not authorized to access ISTM server services. See **Check Preconditions** (section 3.1.4.2.1) for details of the conditions that can result in this return.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the client is not authorized to use iSCSI Software Target services.<130>

3.53.4.2 RegisterNotification (Opnum 11)

Processing of the following error returns SHOULD be performed by client callers of this server method:

ERROR_NOT_READY

A return value of ERROR_NOT_READY indicates that the required iSCSI Software Target services are not available. See **Check Preconditions** (section 3.1.4.2.1) for details of the conditions that can result in this return. Some of the conditions that cause this error are transient, and the operation can be retried at a later time, after a new interface pointer has been obtained through DCOM instantiation.

The caller SHOULD take an implementation-defined action to inform the user of the ISTM client that iSCSI Software Target services are not available.<131>

E_ACCESSDENIED

A return value of E_ACCESSDENIED indicates that the ISTM client is not authorized to access ISTM server services. See **Check Preconditions** (section 3.1.4.2.1) for details of the conditions that can result in this return.

The caller SHOULD take an implementation-defined action to inform the end user of the ISTM client that the client is not authorized to use iSCSI Software Target services.<132>

3.53.5 Timer Events

None.

3.53.6 Other Local Events

None.

3.54 IWTGeneral2 Server Details

The **IWTGeneral2** interface extends **IWTGeneral** (section 3.52) to include information for HA cluster configurations.

3.54.1 Abstract Data Model

For details concerning the abstract data organization for this interface, see the feature management data model (section 3.1.1.1).

3.54.2 Timers

None.

3.54.3 Initialization

Feature manager initialization is specified in section 3.1.3.

3.54.4 Message Processing Events and Sequencing Rules

The **IWTGeneral2** interface includes the following method:

Method	Description
IsClustered	Determines whether the ISTM server target service is running in an HA cluster configuration. Opnum: 15

3.54.4.1 IsClustered (Opnum 15)

The **IsClustered** method determines whether the ISTM server target service is running in an HA cluster configuration.

```
HRESULT IsClustered();
```

Return Values:

The following values are possible:

Return value/code	Description
S_OK 0x00000000	The iSCSI Software Target is running in an HA cluster configuration.
S_FALSE 0x00000001	The iSCSI Software Target is not running in an HA cluster configuration.

A value of S_OK SHOULD NOT be inferred to mean that ISTM server cluster services are currently available.

3.54.5 Timer Events

None.

3.54.6 Other Local Events

None.

4 Protocol Examples

The following sections present a series of examples that represent a common iSCSI target management scenario using the ISTM protocol:

- Creating a Virtual Disk (section 4.1)
- Creating a Target (section 4.2)
- Assigning a Virtual Disk to a Target (section 4.3)
- Enabling an iSCSI Initiator to Connect to a Target (section 4.4)
- Modifying Target Properties (section 4.5)
- Enumerating Existing Virtual Disks (section 4.6)
- Removing a Virtual Disk from a Target (section 4.7)
- Deleting a Virtual Disk (section 4.8)

In all the ISTM protocol examples in this section, the following assumptions have been made:

- A management connection between the ISTM client and ISTM server has already been established.
- The ISTM server machine is not part of an HA cluster.
- The manipulated LUNs are virtual disks in the VHD file format.

4.1 Creating a Virtual Disk

This example describes the ISTM protocol interaction between a client and server for creating a virtual disk.

The following diagram illustrates the sequence of messages that are required for this example. The interfaces used in this example are **ILocalDeviceMgr** (section 3.30) and **IWTDiskMgr** (section 3.46). The contents of the messages are described following the diagram.

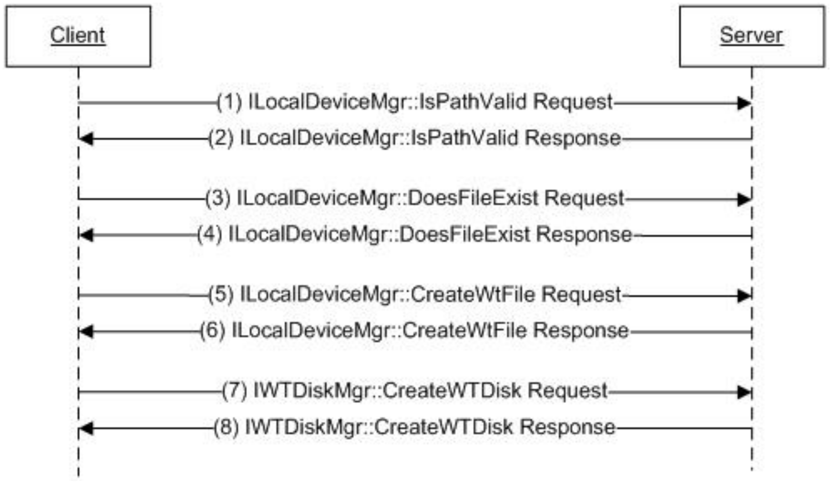

Client
Server
(1) ILocalDeviceMgr::IsPathValid Request
(2) ILocalDeviceMgr::IsPathValid Response
(3) ILocalDeviceMgr::DoesFileExist Request
(4) ILocalDeviceMgr::DoesFileExist Response
(5) ILocalDeviceMgr::CreateWtFile Request
(6) ILocalDeviceMgr::CreateWtFile Response
(7) IWTDiskMgr::CreateWTDisk Request
(8) IWTDiskMgr::CreateWTDisk Response

Figure 17: Sample message sequence for creating a virtual disk

The VHD file that represents the virtual disk is created on a local drive and is located at "C:\iSCSI_target_disks\VirtualDisk1.vhd". Its size is 64 MB, and its description is "Virtual Disk Number 1 – 64 MB".

- ILocalDeviceMgr::IsPathValid request.

The client requests that the server determine whether the given path is valid. The following parameter is included in the IsPathValid request:

Parameter	Value
FilePath	"C:\iSCSI_target_disks\VirtualDisk1.vhd"

The server performs the following checks on the validity of the file path and compatibility with the features of the underlying operating system.

- The path specified by the *FilePath* parameter is within the confines of an existing local hard drive or similar fixed storage device, and is not part of another locally mounted virtual disk.
- The *FilePath* parameter specifies a file, not a directory.
- The VHD file specified by the *FilePath* parameter is not already in use by another iSCSI virtual disk.
- ILocalDeviceMgr::IsPathValid response.

The server returns the value S_OK (0x00000000), indicating that the specified VHD file is free to be used as a virtual disk.

- ILocalDeviceMgr::DoesFileExist request.

The client requests that the server indicate whether the identified file exists on the iSCSI target. The following parameters are included in the **DoesFileExist** method request:

Parameter	Value
FilePath	"C:\iSCSI_target_disks\VirtualDisk1.vhd"

The server determines whether the specified file already exists on a local hard drive or similar fixed storage device.

- ILocalDeviceMgr::DoesFileExist response.

The server returns the value E_FAIL (0x80004005), indicating either that the file does not exist or that it is not possible to establish whether the file exists.

- ILocalDeviceMgr::CreateWtFile request.

The client requests that the server create the specified file on the iSCSI target. The following parameters are included in the **CreateWtFile** method request:

Parameter	Value
FilePath	"C:\iSCSI_target_disks\VirtualDisk1.vhd"

Parameter	Value
hyFileSize	0x0000000004000000
bSparse	0x00 (false)

- ILocalDeviceMgr::CreateWtFile response.

The server returns the value S_OK (0x00000000), indicating success.

- IWTDiskMgr::CreateWTDisk request.

The client requests that the server create a virtual disk on the iSCSI target and map the disk to the identified file. The following parameters are included in the **CreateWTDisk** method request:

Parameter	Value
eDeviceType	0x00000001 (WTDTypeFileBased)
DevicePath	"C:\iSCSI_target_disks\VirtualDisk1.vhd"
Description	"Virtual Disk Number 1 – 64 MB"
bClearPartTable	0x00 (false)
bForceDismount	0x00 (false)

- IWTDiskMgr::CreateWTDisk response.

The server returns the value S_OK (0x00000000), indicating success. In addition, the server returns the **WTDIndex** property (section 3.1.1.5) of the virtual disk; for example, 0x00000004. In a typical C/C++ implementation of the DCOM runtime, that return value is placed at the address specified by the *pWTD* parameter passed to the method.

An alternative sequence of messages would be used if the VHD file had already existed. The following diagram illustrates the sequence of messages that are required for this alternative. The contents of the messages are described following the diagram.

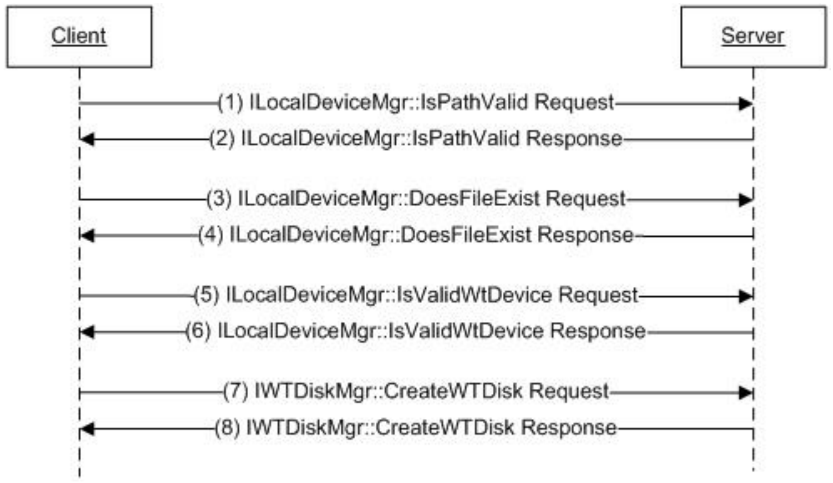

Client
Server
(1) ILocalDeviceMgr::IsPathValid Request
(2) ILocalDeviceMgr::IsPathValid Response
(3) ILocalDeviceMgr::DoesFileExist Request
(4) ILocalDeviceMgr::DoesFileExist Response
(5) ILocalDeviceMgr::IsValidWtDevice Request
(6) ILocalDeviceMgr::IsValidWtDevice Response
(7) IWTDiskMgr::CreateWTDisk Request
(8) IWTDiskMgr::CreateWTDisk Response

Figure 18: Alternative sample message sequence for creating a virtual disk

1. See item 1 in the preceding scenario.

2. See item 2 in the preceding scenario.

3. See item 3 in the preceding scenario.

4. ILocalDeviceMgr::DoesFileExist response.

The server returns the value S_OK (0x00000000), indicating that the file exists.

- ILocalDeviceMgr::IsValidWtDevice request

The client requests that server determine if the given device is valid. The following parameters are included in the **IsValidWtDevice** method request:

Parameter	Value
DevicePath	"C:\iSCSI_target_disks\VirtualDisk1.vhd"

The server determines whether the *DevicePath* parameter specifies an actual VHD file, and whether the type of that VHD file is supported by the iSCSI Software Target.

- ILocalDeviceMgr::IsValidWtDevice response.

The server returns the value S_OK (0x00000000), indicating that the file exists and is a valid VHD file of the correct type.

1. See item 7 in the first message sequence in this section.

2. See item 8 in the first message sequence in this section.

4.2 Creating a Target

This example describes the ISTM protocol interaction between a client and server for creating a target.

The following diagram illustrates the sequence of messages that are required for this example. The interfaces used in this example are **IHostMgr** (section 3.25) and **IHost** (section 3.21). The contents of the messages are described following the diagram.

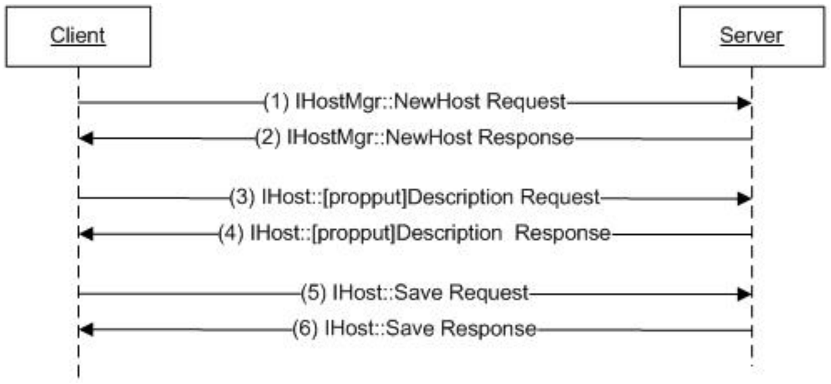

Client
Server
(1) IHostMgr::NewHost Request
(2) IHostMgr::NewHost Response
(3) IHost::[propput]Description Request
(4) IHost::[propput]Description Response
(5) IHost::Save Request
(6) IHost::Save Response

Figure 19: Sample message sequence for creating a target

- IHostMgr::NewHost request

The client requests that the server create a new target. The client sends the name of the target in the following parameter:

Parameter	Value
HostName	"DatabaseTarget"

The server ensures that the name supplied in the *HostName* parameter is not in use by an existing iSCSI target, and it can perform additional validation on the syntax of the name. If the name is available, a placeholder object is created to store the name until the object is explicitly saved.

- IHostMgr::NewHost response,

The server returns the value S_OK (0x00000000), indicating success. In addition, the server returns the interface pointer for the target in the response message. In a typical C/C++ implementation of the DCOM runtime, that return value is placed at the address specified by the *ppNewHost* parameter passed to the method.

- IHost::[propput]Description Request

The client sends this request to the interface pointer for the target obtained in message 2, for updating the **Description** property (section 3.1.1.7) of the target. In this example, the *Description* parameter is specified as follows:

Parameter	Value
Description	"ISTM Test target"

- IHost::[propput]Description Response

The server returns the value S_OK (0x00000000), indicating success. The new **Description** property is retained with the placeholder target created after message 1.

- IHost::Save Request

The client issues a request for saving the target with all of the configured properties to the persistent storage of the ISTM server.

- IHost::Save Response

The server returns the value S_OK (0x00000000), indicating success.

After the ISTM Client has called the **Save** method, the target is retained in the internal collections.

4.3 Assigning a Virtual Disk to a Target

This example describes the ISTM protocol interaction between a client and server for assigning a virtual disk to a target.

The following diagram illustrates the sequence of messages that are required for this example. The interfaces used in this example are **IHostMgr** (section 3.25) and **IHost** (section 3.21). The contents of the messages are described following the diagram.

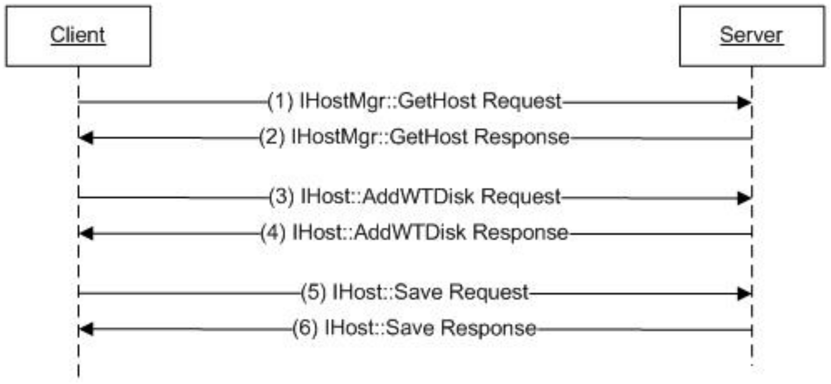

Client
Server
(1) IHostMgr::GetHost Request
(2) IHostMgr::GetHost Response
(3) IHost::AddWTDisk Request
(4) IHost::AddWTDisk Response
(5) IHost::Save Request
(6) IHost::Save Response

Figure 20: Sample message sequence for assigning a virtual disk to a target

In this sequence, it is assumed that a target with the specified name already has been created, as in **Creating a Target** (section 4.2), and that a virtual disk with the specified ID exists.

- IHostMgr::GetHost Request

A client requests a target from the server. The target name is specified in the *HostName* parameter.

Parameter	Value
HostName	"DatabaseTarget"

The ISTM server inspects the collection of existing targets, searching for the specified name. If the name is found, the server returns the target interface pointer.

- IHostMgr::GetHost Response

The server returns the value S_OK (0x00000000), indicating success. In addition, the server returns the interface pointer for the target in the response message. In a typical C/C++ implementation of the DCOM runtime, that return value is placed at the address specified by the *ppHost* parameter passed to the method.

- IHost::AddWTDisk Request

The client requests that the server assign a virtual disk that has the ID 0x00000003, by using the target interface pointer obtained in message 2.

Parameter	Value
nWTD	0x00000003

Upon receiving the request, the server confirms that the specified index maps to an existing LUN, that the LUN is operational and functional, and that admissible limits for LUNs per targets are not exceeded.

- IHost::AddWTDisk Response

The server returns the value S_OK (0x00000000), indicating success.

- IHost::Save Request

The client requests that the server save the target properties.

The server saves the new configuration for the target into persistent storage, and the new configuration is available to subsequent requests for the same target.

- IHost::Save Response

The server returns the value S_OK (0x00000000), indicating success.

4.4 Enabling an Initiator to Connect to a Target

This example describes the ISTM protocol interaction between a client and server for enabling an iSCSI initiator to connect to a target.

The following diagram illustrates the sequence of messages that are required for this example. The interfaces used in this example are **IHostMgr** (section 3.25) and **IHost** (section 3.21). The contents of the messages are described following the diagram.

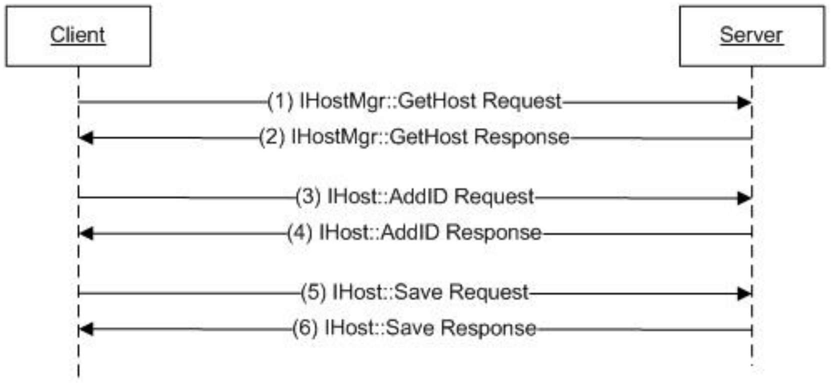

Client
Server
(1) IHostMgr::GetHost Request
(2) IHostMgr::GetHost Response
(3) IHost::AddID Request
(4) IHost::AddID Response
(5) IHost::Save Request
(6) IHost::Save Response

Figure 21: Sample message sequence for enabling an initiator to connect to a target

In this sequence, it is assumed that a target with the specified name already has been created, as in **Creating a Target** (section 4.2), and that a virtual disk with the specified ID exists.

- IHostMgr::GetHost Request

A client requests a target from the server. The target name is specified in the *HostName* parameter.

Parameter	Value
HostName	"DatabaseTarget"

The ISTM server inspects the collection of existing targets, searching for the specified name. If the name is found, the target interface pointer is returned.

- IHostMgr::GetHost Response

The server returns the value S_OK (0x00000000), indicating success. In addition, the server returns the interface pointer for the target in the response message. In a typical C/C++ implementation of the DCOM runtime, that return value is placed at the address specified by the *ppHost* parameter passed to the method.

- IHost::AddID Request

The client requests that the server add an identification method from the target obtained in message 2. The identification method parameter passed in the request message is an **ID** structure (section 2.2.4.7) with fields initialized to the following values.

Parameter	Value
ID::eMethod	0x04 (IDIQN)
ID::szValue	"iqn.1991-05.com.contoso:test-server"

Upon receiving the request, the server validates the syntax of the specified identification method, and if it is a conformant IQN name, it retains the identification method in the collection associated with the target (section 3.1.1.7).

- IHost::AddID Response

The server returns the value S_OK (0x00000000), indicating success.

- IHost::Save Request

The client requests that the server save the target properties.

The server saves the new configuration for the target into persistent storage, and the new configuration is available to subsequent requests for the same target.

- IHost::Save Response

The server returns the value S_OK (0x00000000), indicating success.

4.5 Modifying Target Properties

This example describes the ISTM protocol interaction between a client and server for modifying the properties of a specific target.

The following diagram illustrates the sequence of messages that are required for this example. The interfaces used in this example are **IHostMgr** (section 3.25) and **IHost** (section 3.21). The contents of the messages are described following the diagram.

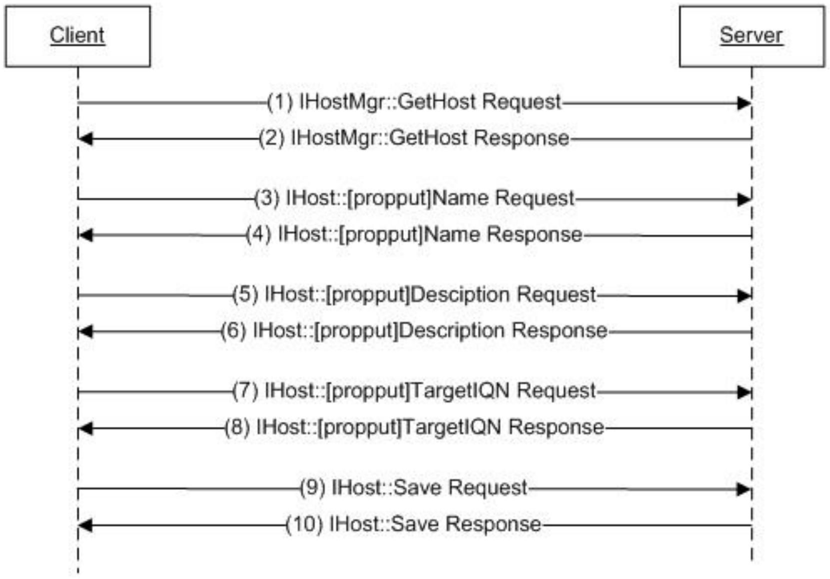

Client
Server
(1) IHostMgr::GetHost Request
(2) IHostMgr::GetHost Response
(3) IHost::[propput]Name Request
(4) IHost::[propput]Name Response
(5) IHost::[propput]Desciption Request
(6) IHost::[propput]Description Response
(7) IHost::[propput]TargetIQN Request
(8) IHost::[propput]TargetIQN Response
(9) IHost::Save Request
(10) IHost::Save Response

Figure 22: Sample message sequence for modifying target properties

In this sequence, it is assumed that a target with the specified name already has been created, as in **Creating a Target** (section 4.2), and that a virtual disk with the specified ID exists.

- IHostMgr::GetHost Request

A client requests a target from the server. The target name is specified in the *HostName* parameter.

Parameter	Value
HostName	"ISTMTarget"

The ISTM server inspects the collection of existing targets, searching for the specified name. If the name is found, the target interface pointer is returned.

- IHostMgr::GetHost Response

The server returns the value S_OK (0x00000000), indicating success. In addition, the server returns the interface pointer for the target in the response message. In a typical C/C++ implementation of the DCOM runtime, that return value is placed at the address specified by the *ppHost* parameter passed to the method.

- IHost:: [propput]Name Request

The client requests that the server update the **Name** property (section 3.1.1.7) on the target obtained in message 2.

Parameter	Value
Name	"DatabaseTarget"

- IHost:: [propput]Name Response

The server returns the value S_OK (0x00000000), indicating success.

- IHost:: [propput]Description Request

The client requests that the server update the **Description** property (section 3.1.1.7) on the target obtained in message 2.

Parameter	Value
Description	"ISTM Test target updated"

- IHostMgr: [propput]Description Response

The server returns the value S_OK (0x00000000), indicating success.

- IHost:: [propput]TargetIQN Request

The client requests that the server update the **TargetIQN** property (section 3.1.1.7) on the target obtained in message 2.

Parameter	Value
TargetIQN	"iqn.1991-5.com.contoso:test-server-istmtarget-target"

The server confirms that the specified string is a valid IQN name and that the name is not already in use by other configured targets.

- IHost:: [propput]TargetIQN Response

The server returns the value S_OK (0x00000000), indicating success.

- IHost:Save Request

The client requests that the server save the target properties.

The server saves the new configuration for the target into persistent storage, and the new configuration is available to subsequent requests for the same target.

- IHost:Save Response

The server returns the value S_OK (0x00000000), indicating success.

4.6 Enumerating Existing Virtual Disks

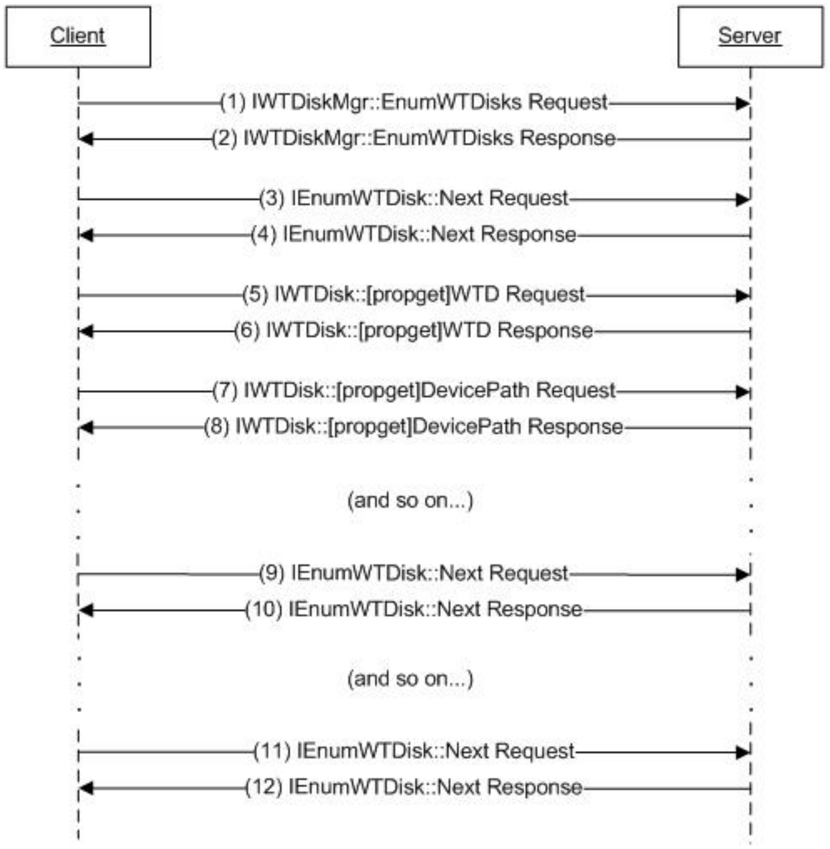
This example describes the ISTM protocol interaction between a client and server when a client requests an enumeration of the existing virtual disks on the iSCSI target.

For this task, the ISTM client does the following:

- Requests an enumerator.
- Using the returned interface pointer for the pool of objects, requests the interface pointer for the first object.
- Using the interface pointer for the first object, requests information about that object.
- Again using the returned interface pointer for the pool of objects, requests the interface pointer for the next object.

- Continues until the required information has been obtained for all objects.

The following diagram illustrates the sequence of messages that are required for this example. The interfaces being used for this example are **IWTDiskMgr** (section 3.46), **IEnumWTDisk** (section 3.19), and **IWTDisk** (section 3.43). The contents of the messages are described following the diagram.



Client
Server
(1) IWTDiskMgr::EnumWTDisks Request
(2) IWTDiskMgr::EnumWTDisks Response
(3) IEnumWTDisk::Next Request
(4) IEnumWTDisk::Next Response
(5) IWTDisk::[propget]WTD Request
(6) IWTDisk::[propget]WTD Response
(7) IWTDisk::[propget]DevicePath Request
(8) IWTDisk::[propget]DevicePath Response
(and so on...)
(9) IEnumWTDisk::Next Request
(10) IEnumWTDisk::Next Response
(and so on...)
(11) IEnumWTDisk::Next Request
(12) IEnumWTDisk::Next Response

Figure 23: Sample message sequence for enumerating virtual disks

- IWTDiskMgr::EnumWTDisks request.

The client requests from the server a pointer to the object interface of the enumerator of virtual disks.

- IWTDiskMgr::EnumWTDisks response.

The server returns the value S_OK (0x00000000), indicating success. In addition, the server returns a pointer to the **IEnumWTDisk** interface (section 3.19), which can be used to obtain pointers to individual disks. In a typical C/C++ implementation of the DCOM runtime, that return value is placed at the address specified by the *ppEnumWTDisk* parameter passed to the method.

- IEnumWTDisk::Next request.

The client requests from the server a pointer to the object interface of the first virtual disk in the pool of virtual disks. The following parameters are included in the request:

Parameter	Value
ulNumElements	0x00000001
pElements	0x00000000 (NULL)

- IEnumWTDisk::Next response.

The server returns the value S_OK (0x00000000), indicating success. In addition, the variable at the address specified in the *pElements* parameter contains an interface pointer to the **IWTDisk** object interface. The client can use this object interface to obtain information about the first virtual disk in the pool of virtual disks. Finally, the variable at the address specified in the *pNumElementsReturned* parameter contains 0x00000001, which is the number of object interface pointers returned.

- IWTDisk::[propget]WTD request.

The client requests from the server the **WTDIndex** property (section 3.1.1.5) of the virtual disk. This method is called by using the interface pointer that was returned in the **Next** method response in message 4.

- IWTDisk::[propget]WTD response.

The server returns the value S_OK (0x00000000), indicating success. In addition, the variable at the address specified in the *pWTD* parameter contains the **WTDIndex** property (section 3.1.1.5) of the virtual disk.

- IWTDisk::[propget]DevicePath request.

The client requests from the server the path of the VHD file of the virtual disk. The **[propget]DevicePath** method is called by using the interface pointer that was returned in the **Next** method response in message 4.

- IWTDisk::[propget]DevicePath response.

The server returns the value S_OK (0x00000000), indicating success. In addition, the variable at the address specified in the *pDevicePath* parameter contains the path of the VHD file; for example, "C:\iSCSI_target_disks\some_virtual_disk.vhd".

- IEnumWTDisk::Next request.

The client requests from the server an interface pointer to the object interface of the second virtual disk in the pool of virtual disks. The following parameters are included in the request:

Parameter	Value
ulNumElements	0x00000001
pElements	0x00000000 (NULL)

- IEnumWTDisk::Next response.

The server returns the value S_OK (0x00000000), indicating success. In addition, the variable at the address specified in the *pElements* parameter contains a pointer to the **IWTDisk** object interface. The client can use this object interface to obtain information about the second virtual disk in the pool of virtual disks. Finally, the variable at the address specified in the *pNumElementsReturned* parameter contains 0x00000001, which is the number of object interface pointers returned.

- IEnumWTDisk::Next request.

The client requests from the server an interface pointer to the object interface of the next virtual disk in the pool of virtual disks. The following parameters are included in the request:

Parameter	Value
ulNumElements	0x00000001
pElements	0x00000000 (NULL)

- IEnumWTDisk::Next response.

The server returns the value S_FALSE (0x00000001), indicating that the request was successfully handled, but that there was no data to return. In addition, the variable at the address specified in the *pElements* parameter contains NULL (0x00000000). Finally, the variable at the address specified in the *pNumElementsReturned* parameter contains 0x00000000, indicating that there are no more virtual disks to be enumerated.

4.7 Removing a Virtual Disk from a Target

This example describes the ISTM protocol interaction between a client and server for deleting a virtual disk from a specific target.

The following diagram illustrates the sequence of messages that are required for this example. The interfaces used in this example are **IHostMgr** (section 3.25) and **IHost** (section 3.21). The contents of the messages are described following the diagram.

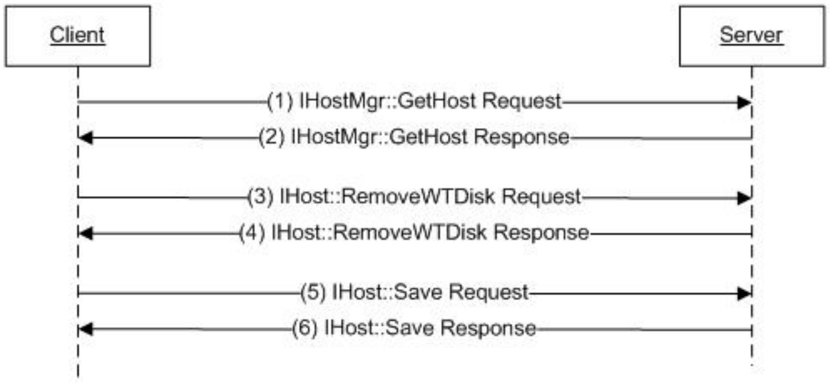

Client
Server
(1) IHostMgr::GetHost Request
(2) IHostMgr::GetHost Response
(3) IHost::RemoveWTDisk Request
(4) IHost::RemoveWTDisk Response
(5) IHost::Save Request
(6) IHost::Save Response

Figure 24: Sample message sequence for removing a virtual disk from a target

In this sequence, it is assumed that a target with the specified name already has been created, as in **Creating a Target** (section 4.2), and that a virtual disk with the specified ID exists.

- IHostMgr::GetHost Request

A client requests a target from the server. The target name is specified in the *HostName* parameter.

Parameter	Value
HostName	"DatabaseTarget"

- IHostMgr::GetHost Response

The server returns the value S_OK (0x00000000), indicating success. In addition, the server returns the interface pointer for the target in the response message. In a typical C/C++ implementation of the DCOM runtime, that return value is placed at the address specified by the *ppHost* parameter passed to the method.

- IHost::RemoveWTDisk Request

The client requests that the server remove the virtual disk with ID 0x00000003 from the target obtained in message 2.

Parameter	Value
nWTD	0x00000003

The server checks that the LUN identification index is present in the collection of LUNs maintained by the LUN manager, and that it was previously assigned to the target. Other possible checks include the following:

- The disk is not in use by an initiator.
- The disk does not have pending I/O operations.
- IHost::RemoveWTDisk Response

The server returns the value S_OK (0x00000000), indicating success.

- IHost:Save Request

The client requests that the server save the target properties.

The server saves the new configuration for the target into persistent storage, and the new configuration is available to subsequent requests for the same target.

- IHost:Save Response

The server returns the value S_OK (0x00000000), indicating success.

4.8 Deleting a Virtual Disk

This example describes the ISTM protocol interaction between a client and server when a client requests the deletion of a virtual disk by the server, followed by the removal of the VHD file that the virtual disk represented.

The following diagram illustrates the sequence of messages that are required for this example. The interfaces used for this example are **IWTDiskMgr** (section 3.46), **IWTDisk** (section 3.43), and **ILocalDeviceMgr** (section 3.30). The sequence is described in the steps that follow the diagram.

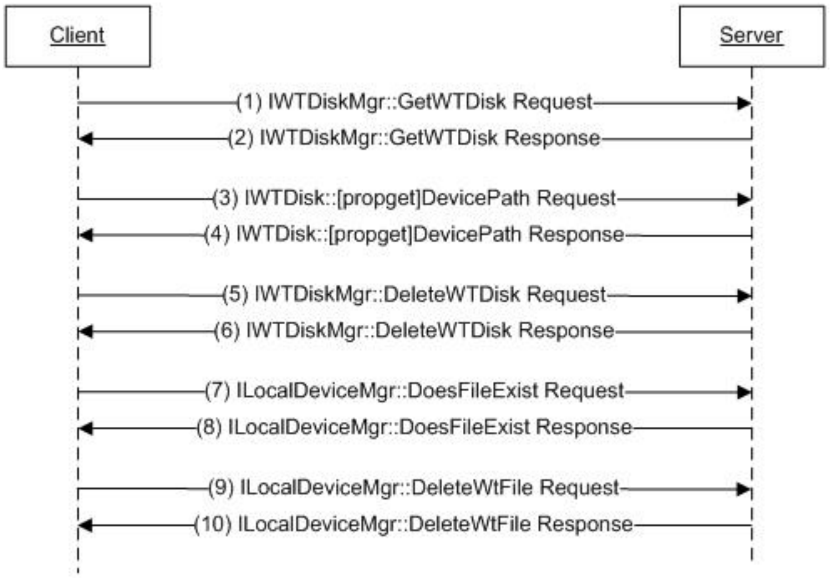

Client
Server
(1) IWTDiskMgr::GetWTDisk Request
(2) IWTDiskMgr::GetWTDisk Response
(3) IWTDisk::[propget]DevicePath Request
(4) IWTDisk::[propget]DevicePath Response
(5) IWTDiskMgr::DeleteWTDisk Request
(6) IWTDiskMgr::DeleteWTDisk Response
(7) ILocalDeviceMgr::DoesFileExist Request
(8) ILocalDeviceMgr::DoesFileExist Response
(9) ILocalDeviceMgr::DeleteWtFile Request
(10) ILocalDeviceMgr::DeleteWtFile Response

Figure 25: Sample message sequence for deleting a virtual disk

In this sequence, it is assumed that the virtual disk to be deleted is not associated with an iSCSI target with one or more active iSCSI initiators.

The **WTDIndex** property (section 3.1.1.5) of the virtual disk is that which was obtained from a call to **CreateWTDisk** as in **Creating a Virtual Disk** (section 4.1). The value of the **WTDIndex** property is 0x00000004 in this example.

- IWTDiskMgr::GetWTDisk request.

The client requests from the server an interface pointer to the object that represents the virtual disk to be deleted. The following parameter is included in the request:

Parameter	Value
nWTD	0x00000004

- IWTDiskMgr::GetWTDisk response.

The server returns the value S_OK (0x00000000), indicating success. In addition, the variable at the address specified in the *ppWTDisk* parameter contains a pointer to the **IWTDisk** object interface. The client can use this object interface to obtain information about the virtual disk identified by the *nWTD* parameter.

- IWTDisk::[propget]DevicePath request.

The client requests from the server the path of the VHD file of the virtual disk. This method is called by using the interface pointer that was returned by the **GetWTDisk** method.

- IWTDisk::[propget]DevicePath response.

The server returns the value S_OK (0x00000000) indicating success. In addition, the variable at the address specified in the *pDevicePath* parameter contains the path of the VHD file. In this example, the path is "C:\iSCSI_target_disks\VirtualDisk1.vhd".

- IWTDiskMgr::DeleteWTDisk request.

The client requests that the server delete the specified virtual disk. The following parameter is included in the request:

Parameter	Value
nWTD	0x00000004

- IWTDiskMgr::DeleteWTDisk response.

The server returns the value S_OK (0x00000000), indicating successful deletion.

Note If the ISTM server has been configured to send unsolicited status notifications, the ISTM client will receive several of these notifications with information about the deletion.

- ILocalDeviceMgr::DoesFileExist request.

The client requests that the server indicate whether the identified file exists on the iSCSI target. The following parameter is included in the request:

Parameter	Value
FilePath	"C:\iSCSI_target_disks\VirtualDisk1.vhd"

- ILocalDeviceMgr::DoesFileExist response.

The server returns the value S_OK (0x00000000), indicating that the file exists.

- ILocalDeviceMgr::DeleteWtFile request.

The client requests that the server delete the specified VHD file. The following parameter is included in the request:

Parameter	Value
FilePath	"C:\iSCSI_target_disks\VirtualDisk1.vhd"

- ILocalDeviceMgr::DeleteWtFile response.

The server returns the value S_OK (0x00000000), indicating that the file was deleted.

5 Security

5.1 Security Considerations for Implementers

The ISTM server requires an **authentication level** of **RPC_C_AUTHN_LEVEL_PKT_PRIVACY** for RPC message protection. See [MS-RPCE] section 2.2.1.1.8 for the meanings of authentication level values**.**

5.2 Index of Security Parameters

None.

6 Appendix A: Full IDL

For ease of implementation the full IDL is provided below, where "ms-dtyp.idl" refers to the IDL found in [MS-DTYP] section 5. The syntax uses the IDL syntax extensions defined in [MS-RPCE] section 2.2.4. For example, as noted in [MS-RPCE] section 2.2.4.9, a pointer_default declaration is not required and pointer_default(unique) is assumed.

```
import "ms-dtyp.idl";
import "ms-oaut.idl";
import "ms-vds.idl";
// ISTM

const int  MAX_RESOURCE_GROUP_NAME = 511;
const int  MAX_IP_ADDRESS_LEN = 127;
typedef enum _DeviceStatus {
        Idle = 0,
        InUse = 1,
        Reverting = 2
}    DeviceStatus;

typedef enum _Type {
        Directory = 0,
        File = 1
}    Type;

typedef enum _IDMethod {
        IDInvalid = -1,
        IDNetBiosName = 0,
        IDDNSName = 1,
        IDIPAddress = 2,
        IDMACAddress = 3,
        IDIQN = 4,
        IDIPv6Address = 5
}    IDMethod;

typedef enum _DayOfWeek {
        Sunday = 0,
        Monday = 1,
        Tuesday = 2,
        Wednesday = 3,
        Thursday = 4,
        Friday = 5,
        Saturday = 6
}    DayOfWeek;

typedef enum _HostStatus {
        HostStatusIdle = 0,
        HostStatusConnected = 1
}    HostStatus;

typedef enum _WTDType {
        WTDTypeVolumeBased = 0,
        WTDTypeFileBased = 1,
        WTDTypeRamBased = 2
}    WTDType;

typedef enum _MountType {
        DoNotMount = 0,
        MountSnapshot = 1,
        MountDirect = 2
}    MountType;

typedef enum _VhdType {
        InvalidVhd = 0,
        FixedVhd = 1,
        DifferencingVhd = 2,
        DynamicVhd = 3
}    VhdType;

typedef enum _SessionType {
        Invalid = 0,
        Discovery = 1,
        Normal = 2
}    SessionType;

typedef enum _AuthenticationMethod {
        None = 0,
        Chap = 1
}    AuthenticationMethod;

typedef struct _LUN_STATUS {
        DWORD  dwLun;
        DeviceStatus  eStatus;
        DWORD  dwProgressPercent;
        DWORD  dwCompletionError;
}    LUN_STATUS;

typedef struct _DISK {
        DWORD  dwIndex;
        WCHAR  szWin32Path[64];
        hyper  hySize;
}    DISK;

typedef struct _DISK2 {
        DWORD  dwIndex;
        WCHAR  szWin32Path[64];
        hyper  hySize;
        WCHAR  szResourceGroup[MAX_RESOURCE_GROUP_NAME + 1];
}    DISK2;

typedef struct _VOLUME {
        DWORD  dwDiskIndex;
        WCHAR  szGuid[64];
        WCHAR  szMountPoint[512];
        WCHAR  szLabel[64];
        WCHAR  szFileSystem[64];
        boolean  bSystemVolume;
        boolean  bPageFile;
        hyper  hyOffset;
        hyper  hySize;
        hyper  hyTotalVolSize;
        BOOL  bValidWTVolume;
        BOOL  bIsContainedInExtendedPartition;
        BOOL  bIsOnDynamicDisk;
        BOOL  bUnallocated;
}    VOLUME;

typedef struct _VOLUME2 {
        DWORD  dwDiskIndex;
        WCHAR  szGuid[64];
        WCHAR  szMountPoint[512];
        WCHAR  szLabel[64];
        WCHAR  szFileSystem[64];
        boolean  bSystemVolume;
        boolean  bPageFile;
        hyper  hyOffset;
        hyper  hySize;
        hyper  hyTotalVolSize;
        hyper  hyFreeSpace;
        BOOL  bValidWTVolume;
        BOOL  bIsContainedInExtendedPartition;
        BOOL  bIsOnDynamicDisk;
        BOOL  bUnallocated;
        BOOL  bSupportSparseFiles;
        BOOL  bIsQuorum;
        WCHAR  szResourceGroup[MAX_RESOURCE_GROUP_NAME + 1];
        hyper  MaxFileSize;
}    VOLUME2;

typedef struct _DIRECTORY_DATA {
        WCHAR  szFileName[512];
        WCHAR  szFullPath[512];
        Type  eType;
}    DIRECTORY_DATA;

typedef struct _PORTAL {
        GUID  Guid;
        WCHAR  szIpAddress[MAX_IP_ADDRESS_LEN + 1];
        DWORD  dwPort;
        BOOL  bListen;
        BOOL  bSecure;
        BOOL  bValid;
}    PORTAL;

typedef struct _SNS_SERVER {
        WCHAR  szName[512];
        BOOL  bDHCPDiscovered;
}    SNS_SERVER;

typedef struct _CACHED_INITIATOR {
        WCHAR  szInitiatorIqn[512];
        hyper  hyLastLoggedOnTime;
}    CACHED_INITIATOR;

typedef struct _ID {
        IDMethod  eMethod;
        WCHAR  szValue[512];
}    ID;

typedef struct _LUN_MAPPING {
        DWORD  dwLun;
        DWORD  dwWTD;
}    LUN_MAPPING;

typedef struct _MOUNT_POINT {
        WCHAR  szPath[512];
}    MOUNT_POINT;
```

```
typedef struct _RESOURCE_GROUP {
        WCHAR  szName[MAX_RESOURCE_GROUP_NAME + 1];
}    RESOURCE_GROUP;

typedef struct _LICENSE {
        boolean  bDemo;
        unsigned int  nNumDemoDaysLeft;
        unsigned int  nNumSeats;
        unsigned int  nFeatures;
        WCHAR  LicenseKeys[10][64];
}    LICENSE;

typedef struct _SNAPSHOT_SCHEDULE_SETTINGS {
        boolean  bEveryday;
        DayOfWeek  eDay;
        hyper  hyStartTime;
}    SNAPSHOT_SCHEDULE_SETTINGS;

typedef struct _DRIVE_INFO {
        WCHAR  szDrive[20];
        hyper  hySize;
        hyper  hyFreeSpace;
        boolean  bSupportSparseFiles;
}    DRIVE_INFO;

typedef struct _SESSION {
        WCHAR  InitiatorIQN[256];
        WCHAR  HostTargetIQN[256];
}    SESSION;

typedef struct _SESSION2 {
        WCHAR  InitiatorIQN[256];
        WCHAR  HostTargetIQN[256];
        WCHAR  HostName[512];
        WORD  TSIH;
        DWORDLONG  ISID;
        SessionType  eType;
        AuthenticationMethod  eAuthMethod;
}    SESSION2;

typedef struct _CONNECTION {
        DWORD  CID;
        WCHAR  InitiatorIpAddress[MAX_IP_ADDRESS_LEN + 1];
        DWORD  InitiatorPort;
        WCHAR  TargetIpAddress[MAX_IP_ADDRESS_LEN + 1];
        DWORD  TargetPort;
        boolean  HeaderDigestEnabled;
        boolean  DataDigestEnabled;
}    CONNECTION;

typedef struct _LOGICAL_UNIT_IDENTIFIER {
        BYTE  NaaType_CompanyIdHigh;
        BYTE  CompanyIdMid[2];
        BYTE  CompanyIdLow_VsidHigh;
        DWORD  VsidLow;
        hyper  VsidExtension;
}    LOGICAL_UNIT_IDENTIFIER;
```

```
typedef struct _WINTARGET_DISK_DEVICE_IDENTIFIER {
        BYTE  szSignature[10];
        BYTE  szServerName[150];
        DWORD  dwLun;
        GUID  SnapshotId;
}    WINTARGET_DISK_DEVICE_IDENTIFIER;

[uuid(312cc019-d5cd-4ca7-8c10-9e0a661f147e)]
[object]
[pointer_default(unique)]
interface IStatusNotify : IUnknown {
        HRESULT LunListChanged(
                [in]unsigned long  lCookie);

        HRESULT LunStatusUpdate(
                [in]unsigned long  lCookie,
                [in]LUN_STATUS * pLunStatus);
};

[uuid(bb39e296-ad26-42c5-9890-5325333bb11e)]
[object]
[pointer_default(unique)]
interface IEnumDisk : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]DISK * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumDisk ** ppNewEnum);
};

[uuid(a5ecfc73-0013-4a9e-951c-59bf9735fdda)]
[object]
[pointer_default(unique)]
interface IEnumDisk2 : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]DISK2 * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumDisk2 ** ppNewEnum);
};

[uuid(81fe3594-2495-4c91-95bb-eb5785614ec7)]
[object]
[pointer_default(unique)]
interface IEnumVolume : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]VOLUME * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumVolume ** ppNewEnum);
};

[uuid(8c58f6b3-4736-432a-891d-389de3505c7c)]
[object]
[pointer_default(unique)]
interface IEnumVolume2 : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]VOLUME2 * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumVolume2 ** ppNewEnum);
};

[uuid(28bc8d5e-ca4b-4f54-973c-ed9622d2b3ac)]
[object]
[pointer_default(unique)]
interface IDirectoryEnum : IUnknown {
        HRESULT GetDriveString(
                [out]BSTR * pbstrDriveString);

        HRESULT FindFirst(
                [in]BSTR  bstrSearchPath,
                [in, out]DIRECTORY_DATA * pDirectoryData);

        HRESULT FindNext(
                [in, out]DIRECTORY_DATA * pDirectoryData);

        HRESULT FindClose();
};

[uuid(1995785d-2a1e-492f-8923-e621eaca39d9)]
[object]
[pointer_default(unique)]
interface IEnumPortal : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]PORTAL * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumPortal ** ppNewEnum);
};

[uuid(e2842c88-07c3-4eb0-b1a9-d3d95e76fef2)]
[object]
[pointer_default(unique)]
interface IEnumSnsServer : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]SNS_SERVER * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumSnsServer ** ppNewEnum);
};

[uuid(f093fe3d-8131-4b73-a742-ef54c20b337b)]
[object]
[pointer_default(unique)]
interface IEnumCachedInitiator : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]CACHED_INITIATOR * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumCachedInitiator ** ppNewEnum);
};

[uuid(345b026b-5802-4e38-ac75-795e08b0b83f)]
[object]
[pointer_default(unique)]
interface IEnumIDMethod : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]ID * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumIDMethod ** ppNewEnum);
};

[uuid(1396de6f-a794-4b11-b93f-6b69a5b47bae)]
[object]
[pointer_default(unique)]
interface IEnumWTDiskLunMapping : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]LUN_MAPPING * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumWTDiskLunMapping ** ppNewEnum);
};

[uuid(b0076fec-a921-4034-a8ba-090bc6d03bde)]
[object]
[pointer_default(unique)]
interface IHost : IUnknown {
        [propget]HRESULT Guid(
                [out][retval]
                GUID * pGuid);

        [propget]HRESULT Enable(
                [out][retval]
                boolean * pEnable);

        [propput]HRESULT Enable(
                [in]boolean  bEnable);

        [propget]HRESULT Name(
                [out][retval]
                BSTR * pName);

        [propput]HRESULT Name(
                [in]BSTR  Name);

        [propget]HRESULT Description(
                [out][retval]
                BSTR * pDescription);

        [propput]HRESULT Description(
                [in]BSTR  Description);

        [propget]HRESULT TargetIQN(
                [out][retval]
                BSTR * pTargetIQN);

        [propput]HRESULT TargetIQN(
                [in]BSTR  TargetIQN);

        [propget]HRESULT Status(
                [out][retval]
                HostStatus * pStatus);

        [propput]HRESULT Status(
                [in]HostStatus  eStatus);

        [propget]HRESULT LastLogIn(
                [out][retval]
                hyper * pTimeStamp);

        [propput]HRESULT LastLogIn(
                [in]hyper  hyTimeStamp);

        [propget]HRESULT EnableCHAP(
                [out][retval]
                boolean * pEnable);

        [propput]HRESULT EnableCHAP(
                [in]boolean  bEnable);

        [propget]HRESULT CHAPUserName(
                [out][retval]
                BSTR * pUserName);

        [propput]HRESULT CHAPUserName(
                [in]BSTR  UserName);

        [propget]HRESULT CHAPSecret(
                [out][retval]
                BSTR * pSecret);

        [propput]HRESULT CHAPSecret(
                [in]BSTR  Secret);

        [propget]HRESULT EnableReverseCHAP(
                [out][retval]
                boolean * pEnable);

        [propput]HRESULT EnableReverseCHAP(
                [in]boolean  bEnable);

        [propget]HRESULT ReverseCHAPUserName(
                [out][retval]
                BSTR * pUserName);

        [propput]HRESULT ReverseCHAPUserName(
                [in]BSTR  UserName);

        [propget]HRESULT ReverseCHAPSecret(
                [out][retval]
                BSTR * pSecret);

        [propput]HRESULT ReverseCHAPSecret(
                [in]BSTR  Secret);

        [propget]HRESULT TargetMaxRecvDataSegmentLength(
                [out][retval]
                unsigned long * pLength);

        [propput]HRESULT TargetMaxRecvDataSegmentLength(
                [in]unsigned long  nLength);

        [propget]HRESULT TargetFirstBurstLength(
                [out][retval]
                unsigned long * pLength);

        [propput]HRESULT TargetFirstBurstLength(
                [in]unsigned long  nLength);

        [propget]HRESULT TargetMaxBurstLength(
                [out][retval]
                unsigned long * pLength);

        [propput]HRESULT TargetMaxBurstLength(
                [in]unsigned long  nLength);

        [propget]HRESULT NumRecvBuffers(
                [out][retval]
                unsigned long * pNumBuffers);

        [propput]HRESULT NumRecvBuffers(
                [in]unsigned long  nNumBuffers);

        [propget]HRESULT EnforceIdleTimeoutDetection(
                [out][retval]
                boolean * pEnable);

        [propput]HRESULT EnforceIdleTimeoutDetection(
                [in]boolean  bEnable);

        [propget]HRESULT ListenAllNetworkAdapters(
                [out][retval]
                boolean * pListenAll);

        [propput]HRESULT ListenAllNetworkAdapters(
                [in]boolean  bListenAll);

        [propget]HRESULT Processor(
                [out][retval]
                unsigned long * pProcessor);

        [propput]HRESULT Processor(
                [in]unsigned long  nProcessor);

        HRESULT EnumIDs(
                [out]IEnumIDMethod ** ppEnumIDMethod);

        HRESULT AddID(
                [in]ID * pIDMethod);

        HRESULT RemoveID(
                [in]ID * pIDMethod);

        HRESULT RemoveAllIDs();

        HRESULT EnumWTDiskLunMappings(
                [out]IEnumWTDiskLunMapping ** ppEnumWTDiskLunMapping);

        HRESULT AddWTDisk(
                [in]unsigned long  nWTD);

        HRESULT RemoveWTDisk(
                [in]unsigned long  nWTD);

        HRESULT RemoveAllWTDisks();

        HRESULT SetWTDiskLunMapping(
                [in]unsigned long  nWTD,
                [in]unsigned long  nNewLun);

        HRESULT EnumPortals(
                [out]IEnumPortal ** ppEnumPortal);

        HRESULT AddPortal(
                [in]BSTR  IpAddress);

        HRESULT RemovePortal(
                [in]BSTR  IpAddress);

        HRESULT RemoveAllPortals();

        HRESULT SetCHAPInfoByBlob(
                [in]unsigned long  nBlobSize,
                [in][size_is(nBlobSize)]
                char  Blob[]);

        HRESULT Save();

        HRESULT ValidateChapSecret(
                [in]BSTR  Secret);

        HRESULT ValidateReverseChapSecret(
                [in]BSTR  Secret);

        HRESULT CheckTargetRedirect(
                [in]BSTR  LocalConnectAddr,
                [out]BSTR * pRedirectAddr);
};

[uuid(100da538-3f4a-45ab-b852-709148152789)]
[object]
[pointer_default(unique)]
interface IEnumLMMountPoint : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]MOUNT_POINT * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumLMMountPoint ** ppNewEnum);
};

[uuid(866a78bc-a2fb-4ac4-94d5-db3041b4ed75)]
```

```
[object]
[pointer_default(unique)]
interface IWTDisk : IUnknown {
        [propget]HRESULT Guid(
                [out][retval]
                GUID * pGuid);

        [propget]HRESULT SerialNumber(
                [out][retval]
                BSTR * pSerialNumber);

        [propget]HRESULT WTD(
                [out][retval]
                unsigned long * pWTD);

        [propput]HRESULT WTD(
                [in]unsigned long  nWTD);

        [propget]HRESULT Type(
                [out][retval]
                WTDType * pType);

        [propput]HRESULT Type(
                [in]WTDType  eType);

        [propget]HRESULT Flags(
                [out][retval]
                unsigned long * pFlags);

        [propput]HRESULT Flags(
                [in]unsigned long  nFlags);

        [propget]HRESULT Status(
                [out][retval]
                DeviceStatus * pStatus);

        [propput]HRESULT Status(
                [in]DeviceStatus  eStatus);

        [propget]HRESULT Assigned(
                [out][retval]
                boolean * pAssigned);

        [propput]HRESULT Assigned(
                [in]boolean  bAssigned);

        [propget]HRESULT Description(
                [out][retval]
                BSTR * pDescription);

        [propput]HRESULT Description(
                [in]BSTR  Description);

        [propget]HRESULT DevicePath(
                [out][retval]
                BSTR * pDevicePath);

        [propput]HRESULT DevicePath(
                [in]BSTR  DevicePath);
        [propget]HRESULT Size(
                [out][retval]
                hyper * pSize);

        [propget]HRESULT SnapshotStoragePath(
                [out][retval]
                BSTR * pSnapshotStoragePath);

        [propput]HRESULT SnapshotStoragePath(
                [in]BSTR  SnapshotStoragePath);

        [propget]HRESULT SnapshotStorageSize(
                [out][retval]
                hyper * pSize);

        [propput]HRESULT SnapshotStorageSize(
                [in]hyper  nSize);

        [propget]HRESULT SnapshotStorageUsedSize(
                [out][retval]
                hyper * pSize);

        [propget]HRESULT LMType(
                [out][retval]
                MountType * pType);

        [propput]HRESULT LMType(
                [in]MountType  eType);

        [propget]HRESULT LMStatus(
                [out][retval]
                MountType * pStatus);

        [propget]HRESULT LMTimeStamp(
                [out][retval]
                hyper * pTimeStamp);

        [propget]HRESULT LMWTManageMountPoint(
                [out][retval]
                boolean * pManage);

        [propput]HRESULT LMWTManageMountPoint(
                [in]boolean  bManage);

        [propget]HRESULT LMMountPoint(
                [out][retval]
                BSTR * pPath);

        [propput]HRESULT LMMountPoint(
                [in]BSTR  Path);

        [propget]HRESULT LMCurrMountDeviceId(
                [out][retval]
                BSTR * pDeviceId);

        [propput]HRESULT LMCurrMountDeviceId(
                [in]BSTR  DeviceId);
        [propget]HRESULT LMSnapshotId(
                [out][retval]
                GUID * pLMSnapshotId);

        HRESULT Save();

        HRESULT DVMountDirect();

        HRESULT DVDismount();

        HRESULT EnumLMMountPoints(
                [out]IEnumLMMountPoint ** ppEnumLMMountPoint);

        HRESULT GetVdsLunInfo(
                [in, out]VDS_LUN_INFORMATION * pVdsLunInfo);

        HRESULT GetDeviceVolumeGuid(
                [out]GUID * pGuid);
};

[uuid(883343f1-ceed-4e3a-8c1b-f0dadfce281e)]
[object]
[pointer_default(unique)]
interface ISnapshot : IUnknown {
        [propget]HRESULT Id(
                [out][retval]
                GUID * pId);

        [propput]HRESULT Id(
                [in]GUID  Id);

        [propput]HRESULT VssSnapshotId(
                [in]GUID  Id);

        [propput]HRESULT VssSnapshotSetId(
                [in]GUID  Id);

        [propget]HRESULT OrigWTD(
                [out][retval]
                unsigned long * pWTD);

        [propput]HRESULT OrigWTD(
                [in]unsigned long  nWTD);

        [propget]HRESULT TimeStamp(
                [out][retval]
                hyper * pTimeStamp);

        [propget]HRESULT ExportedWTD(
                [out][retval]
                unsigned long * pWTD);

        [propput]HRESULT ExportedWTD(
                [in]unsigned long  nWTD);

        [propget]HRESULT Persistent(
                [out][retval]
                boolean * pPersistent);
        [propget]HRESULT Status(
                [out][retval]
                unsigned long * pStatus);

        [propput]HRESULT Status(
                [in]unsigned long  nStatus);

        HRESULT Save();

        HRESULT Rollback();

        HRESULT AbortRollback();

        HRESULT ExportAsWTDisk(
                [out]unsigned long * pWTD,
                [in, out]VDS_LUN_INFORMATION * pVdsLunInfo);

        HRESULT GetVdsLunInfo(
                [in, out]VDS_LUN_INFORMATION * pVdsLunInfo);

        HRESULT DVMount();

        HRESULT DVDismount();
};

[uuid(640038f1-d626-40d8-b52b-09660601d045)]
[object]
[pointer_default(unique)]
interface IEnumResourceGroup : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]RESOURCE_GROUP * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumResourceGroup ** ppNewEnum);
};

[uuid(d71b2cae-33e8-4567-ae96-3ccf31620be2)]
[object]
[pointer_default(unique)]
interface IWTGeneral : IUnknown {
        [propget]HRESULT Guid(
                [out][retval]
                GUID * pGuid);

        [propget]HRESULT Name(
                [out][retval]
                BSTR * pName);

        [propput]HRESULT Name(
                [in]BSTR  Name);

        [propget]HRESULT License(
                [out][retval]
                LICENSE * pLicense);

        [propget]HRESULT WTCliPath(
                [out]BSTR * pWtCliPath);

        HRESULT CheckAccess();

        HRESULT AddLicense(
                [in]BSTR  License);

        HRESULT IsSnapshotsSupported();

        HRESULT RegisterNotification(
                [in]IStatusNotify * pStatusNotify,
                [in]unsigned long  lCookie);

        HRESULT UnregisterNotification(
                [in]IStatusNotify * pStatusNotify);

        HRESULT DVGetSnapshotSchedule(
                [in, out]SNAPSHOT_SCHEDULE_SETTINGS * pSchedule);

        HRESULT DVSetSnapshotSchedule(
                [in]SNAPSHOT_SCHEDULE_SETTINGS  Schedule);
};

[uuid(3c73848a-a679-40c5-b101-c963e67f9949)]
[object]
[pointer_default(unique)]
interface IWTGeneral2 : IWTGeneral {
        HRESULT IsClustered();
};

[uuid(8ad608a4-6c16-4405-8879-b27910a68995)]
[object]
[pointer_default(unique)]
interface ILocalDeviceMgr : IUnknown {
        HRESULT EnumDisks(
                [out]IEnumDisk ** ppEnumDisk);

        HRESULT EnumVolumes(
                [out]IEnumVolume ** ppEnumVolume);

        HRESULT CreateVolume(
                [in]unsigned long  nDiskIndex,
                [in]hyper  hyOffset,
                [in]hyper  hyLength,
                [in]unsigned long  nVolumeType,
                [out]BSTR * DevicePath);

        HRESULT IsValidWtDevice(
                [in]BSTR  DevicePath);

        HRESULT IsDiskPartAvailable(
                [out]boolean * bAvail);

        HRESULT CreateWtFile(
                [in]BSTR  FilePath,
                [in]hyper  hyFileSize,
                [in]boolean  bSparse);

        HRESULT GrowWtFile(
                [in]BSTR  FilePath,
                [in]hyper  hyAdditionalSize);

        HRESULT DeleteWtFile(
                [in]BSTR  FilePath);

        HRESULT IsPathValid(
                [in]BSTR  FilePath);

        HRESULT IsPathNtfs(
                [in]BSTR  Path);

        HRESULT DoesFileExist(
                [in]BSTR  FilePath);

        HRESULT GetDriveInfo(
                [in]BSTR  Drive,
                [in, out]DRIVE_INFO * pDriveInfo);

        HRESULT EnumDirectory(
                [out]IDirectoryEnum ** ppDirectoryEnum);

        HRESULT EnumPortals(
                [out]IEnumPortal ** ppEnumPortal);

        HRESULT GetPortal(
                [in]BSTR  IpAddress,
                [out]PORTAL * pPortal);

        HRESULT SetPortalConfig(
                [in]BSTR  IpAddress,
                [in]unsigned long  nPort,
                [in]boolean  bUse);

        HRESULT DeviceManagerRescan(
                [in]boolean  bSynchronous);

        HRESULT GetProcessors(
                [out]unsigned long * pProcessors);
};

[uuid(b0d1ac4b-f87a-49b2-938f-d439248575b2)]
[object]
[pointer_default(unique)]
interface ILocalDeviceMgr2 : ILocalDeviceMgr {
        HRESULT EnumDisks2(
                [out]IEnumDisk2 ** ppEnumDisk);

        HRESULT EnumVolumes2(
                [out]IEnumVolume2 ** ppEnumVolume);

        HRESULT GetVolumeInfo(
                [in]BSTR  VolumePath,
                [in, out]VOLUME2 * pVolumeInfo);
        HRESULT EnumResourceGroup(
                [out]IEnumResourceGroup ** ppEnumResourceGroup);

        HRESULT EnumLocalResourceGroup(
                [out]IEnumResourceGroup ** ppEnumResourceGroup);

        HRESULT GetResourceGroupGuid(
                [in]BSTR  ResourceGroup,
                [out]GUID * pGuid);

        HRESULT GetResourceGroupFromPath(
                [in]BSTR  FilePath,
                [out]BSTR * pResourceGroup);

        HRESULT RenameResourceGroup(
                [in]BSTR  ResourceGroup,
                [in]BSTR  NewResourceGroup);

        HRESULT EnumLocalPortals(
                [in]BSTR  ResourceGroup,
                [out]IEnumPortal ** ppEnumPortal);

        HRESULT GetResourceGroupOwner(
                [in]BSTR  ResourceGroup,
                [out]BSTR * pOwner);
};

[uuid(e645744b-cae5-4712-acaf-13057f7195af)]
[object]
[pointer_default(unique)]
interface ILocalDeviceMgr3 : ILocalDeviceMgr2 {
        HRESULT CreateDiffWtFile(
                [in]BSTR  FilePath,
                [in]BSTR  ParentPath);
};

[uuid(1b1c4d1c-abc4-4d3a-8c22-547fba3aa8a0)]
[object]
[pointer_default(unique)]
interface ISnsMgr : IUnknown {
        HRESULT EnumSnsServers(
                [out]IEnumSnsServer ** ppEnumSnsServer);

        HRESULT AddSnsServer(
                [in]BSTR  SnsServerName);

        HRESULT RemoveSnsServer(
                [in]BSTR  SnsServerName);

        HRESULT EnumCachedInitiators(
                [out]IEnumCachedInitiator ** ppEnumCachedInitiator);
};

[uuid(e141fd54-b79e-4938-a6bb-d523c3d49ff1)]
[object]
[pointer_default(unique)]
interface IEnumHost : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]IHost ** pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumHost ** ppNewEnum);
};

[uuid(dd6f0a28-248f-4dd3-afe9-71aed8f685c4)]
[object]
[pointer_default(unique)]
interface IHostMgr2 : IHostMgr {
        HRESULT NewHost2(
                [in]BSTR  HostName,
                [in]BSTR  ResourceGroup,
                [out]IHost ** ppNewHost);

        HRESULT NewHostNonPersistent(
                [in]BSTR  HostName,
                [in]BSTR  ResourceName,
                [out]IHost ** ppNewHost);

        HRESULT OpenHost(
                [in]IUnknown * pStream,
                [in]BSTR  ResourceName,
                [out]IHost ** ppHost);

        HRESULT CloseHost(
                [in]BSTR  HostName);
};

[uuid(01454b97-c6a5-4685-bea8-9779c88ab990)]
[object]
[pointer_default(unique)]
interface IHostMgr3 : IHostMgr2 {
        HRESULT EnumAllClusterHosts(
                [out]IEnumHost ** EnumHost);
};

[uuid(b06a64e3-814e-4ff9-afac-597ad32517c7)]
[object]
[pointer_default(unique)]
interface IHostMgr : IUnknown {
        HRESULT EnumHosts(
                [out]IEnumHost ** ppEnumHost);

        HRESULT NewHost(
                [in]BSTR  HostName,
                [out]IHost ** ppNewHost);

        HRESULT GetHost(
                [in]BSTR  HostName,
                [out]IHost ** ppHost);

        HRESULT GetHostByTargetIQN(
                [in]BSTR  TargetIQN,
                [out]IHost ** ppHost);

        HRESULT DeleteHost(
                [in]BSTR  HostName);
};

[uuid(442931d5-e522-4e64-a181-74e98a4e1748)]
[object]
[pointer_default(unique)]
interface IEnumSession2 : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]SESSION2 * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumSession2 ** ppNewEnum);
};

[uuid(fe7f99f9-1dfb-4afb-9d00-6a8dd0aabf2c)]
[object]
[pointer_default(unique)]
interface IHost3 : IHost2 {
        [propget]HRESULT ResourceState(
                [out][retval]
                long * ResourceState);

        [propput]HRESULT ResourceState(
                [in]long  ResourceState);

};

[uuid(56e65ea5-cdff-4391-ba76-006e42c2d746)]
[object]
[pointer_default(unique)]
interface IEnumWTDisk : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]IWTDisk ** pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumWTDisk ** ppNewEnum);
};

[uuid(348a0821-69bb-4889-a101-6a9bde6fa720)]
[object]
[pointer_default(unique)]
```

```
interface IWTDisk2 : IWTDisk {
        [propget]HRESULT Enable(
                [out][retval]
                boolean * pEnable);

        [propput]HRESULT Enable(
                [in]boolean  bEnable);

        [propget]HRESULT ResourceName(
                [out][retval]
                BSTR * pResourceName);

        [propput]HRESULT ResourceName(
                [in]BSTR  ResourceName);

        [propget]HRESULT ResourceGroupName(
                [out][retval]
                BSTR * pResourceGroup);

        [propput]HRESULT ResourceGroupName(
                [in]BSTR  ResourceGroup);

        HRESULT IsAlive();

        HRESULT SaveToStream(
                [in]IUnknown * pStream);
};

[uuid(1822a95e-1c2b-4d02-ab25-cc116dd9dbde)]
[object]
[pointer_default(unique)]
interface IWTDisk3 : IWTDisk2 {
        [propget]HRESULT VhdType(
                [out][retval]
                VhdType * Type);

        [propget]HRESULT ParentPath(
                [out][retval]
                BSTR * ParentPath);

        [propget]HRESULT AllocatedSize(
                [out][retval]
                hyper * Size);

        [propget]HRESULT CacheParent(
                [out][retval]
                boolean * CacheParent);

        [propput]HRESULT CacheParent(
                [in]boolean  CacheParent);

        [propget]HRESULT ResourceState(
                [out][retval]
                long * ResourceState);

        [propput]HRESULT ResourceState(
                [in]long  ResourceState);

        HRESULT GetLunStatus(
                [out]LUN_STATUS * Status);
};

[uuid(52ba97e7-9364-4134-b9cb-f8415213bdd8)]
[object]
[pointer_default(unique)]
interface IWTDiskMgr : IUnknown {
        HRESULT EnumWTDisks(
                [out]IEnumWTDisk ** ppEnumWTDisk);

        HRESULT CreateWTDisk(
                [in]WTDType  eDeviceType,
                [in]BSTR  DevicePath,
                [in]BSTR  Description,
                [in]boolean  bClearPartTable,
                [in]boolean  bForceDismount,
                [out]unsigned long * pWTD);

        HRESULT CreateShadowWTDisk(
                [in]WTDType  eDeviceType,
                [in]BSTR  DevicePath,
                [in]BSTR  Description,
                [in]GUID  SnapshotId,
                [out]unsigned long * pWTD);

        HRESULT DeleteWTDisk(
                [in]unsigned long  nWTD);

        HRESULT GetWTDisk(
                [in]unsigned long  nWTD,
                [out]IWTDisk ** ppWTDisk);

        HRESULT EnumWTDisksOnSameVolumeByPath(
                [in]BSTR  Path,
                [out]IEnumWTDisk ** ppEnumWTDisk);

        HRESULT EnumWTDisksOnSameVolume(
                [in]unsigned long  nWTD,
                [out]IEnumWTDisk ** ppEnumWTDisk);

        HRESULT DVMountSelectedWTDisksSnapshotNow();

        HRESULT ImportWTDisk(
                [in]IWTDisk * pWTDisk);
};

[uuid(592381e5-8d3c-42e9-b7de-4e77a1f75ae4)]
[object]
[pointer_default(unique)]
interface IWTDiskMgr2 : IWTDiskMgr {
        HRESULT CreateWTDiskNonPersistent(
                [in]unsigned long  nWTD,
                [in]WTDType  eDeviceType,
                [in]BSTR  DevicePath,
                [in]BSTR  Description,
                [in]boolean  bClearPartTable,
                [in]boolean  bForceDismount,
                [in]BSTR  ResourceName,
                [out]IWTDisk ** ppWTDisk);
        HRESULT CreateShadowWTDiskNonPersistent(
                [in]unsigned long  nWTD,
                [in]WTDType  eDeviceType,
                [in]BSTR  DevicePath,
                [in]BSTR  Description,
                [in]GUID  SnapshotId,
                [in]BSTR  ResourceName,
                [out]IWTDisk ** ppWTDisk);

        HRESULT OpenWTDisk(
                [in]IUnknown * pStream,
                [in]BSTR  ResourceName,
                [out]IWTDisk ** ppWTDisk);

        HRESULT CloseWTDisk(
                [in]unsigned long  nWTD);
};

[uuid(d6bd6d63-e8cb-4905-ab34-8a278c93197a)]
[object]
[pointer_default(unique)]
interface IWTDiskMgr3 : IWTDiskMgr2 {
        HRESULT ClusterNotifyWTDiskDeletion(
                [in]unsigned long  nWTD);
};

[uuid(703e6b03-7ad1-4ded-ba0d-e90496ebc5de)]
[object]
[pointer_default(unique)]
interface IWTDiskMgr4 : IWTDiskMgr3 {
        HRESULT EnumAllClusterWTDisks(
                [out]IEnumWTDisk ** EnumWTDisk);

        HRESULT ClusterNotifyWTDiskDeletion2(
                [in]unsigned long  nWTD,
                [in] boolean bShadowDisk);
};

[uuid(66c9b082-7794-4948-839a-d8a5a616378f)]
[object]
[pointer_default(unique)]
interface IEnumSnapshot : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]ISnapshot ** pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumSnapshot ** ppNewEnum);
};

[uuid(4e65a71e-4ede-4886-be67-3c90a08d1f29)]
[object]
```

```
[pointer_default(unique)]
interface ISnapshotMgr : IUnknown {
        HRESULT EnumSnapshots(
                [out]IEnumSnapshot ** ppEnumSnapshot);

        HRESULT GetSnapshot(
                [in]GUID  SnapshotId,
                [out]ISnapshot ** ppSnapshot);

        HRESULT PrepareCreateSnapshot(
                [in]unsigned long  nWTD,
                [in, out]GUID * pSnapshotId);

        HRESULT DoCreateSnapshot(
                [in]GUID  SnapshotId);

        HRESULT DoEndSnapshot(
                [in]GUID  SnapshotId);

        HRESULT DeleteSnapshot(
                [in]GUID  SnapshotId);

        HRESULT ImportSnapshot(
                [in]ISnapshot * pSnapshot);
};

[uuid(40cc8569-6d23-4005-9958-e37f08ae192b)]
[object]
[pointer_default(unique)]
interface IEnumSession : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]SESSION * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumSession ** ppNewEnum);
};

[uuid(b4fa8e86-2517-4a88-bd67-75447219eee4)]
[object]
[pointer_default(unique)]
interface IHost2 : IHost {
        [propget]HRESULT ResourceName(
                [out][retval]
                BSTR * pResourceName);

        [propput]HRESULT ResourceName(
                [in]BSTR  ResourceName);

        [propget]HRESULT ResourceGroupName(
                [out][retval]
                BSTR * pResourceGroup);
        [propput]HRESULT ResourceGroupName(
                [in]BSTR  ResourceGroup);

        HRESULT SaveToStream(
                [in]IUnknown * pStream);

        HRESULT EnumSessions(
                [out]IEnumSession2 ** ppEnumSession);
};

[uuid(6aea6b26-0680-411d-8877-a148df3087d5)]
[object]
[pointer_default(unique)]
interface IEnumConnection : IUnknown {
        HRESULT Next(
                [in]ULONG  ulNumElements,
                [in, out]CONNECTION * pElements,
                [out]ULONG * pNumElementsReturned);

        HRESULT Skip(
                [in]ULONG  ulNumElements);

        HRESULT Reset();

        HRESULT Clone(
                [in]IEnumConnection ** ppNewEnum);
};

[uuid(8d7ae740-b9c5-49fc-a11e-89171907cb86)]
[object]
[pointer_default(unique)]
interface ISessionManager : IUnknown {
        HRESULT EnumSessions(
                [out]IEnumSession ** ppEnumSession);
};

[uuid(c10a76d8-1fe4-4c2f-b70d-665265215259)]
[object]
[pointer_default(unique)]
interface ISessionManager2 : ISessionManager {
        HRESULT EnumSessions2(
                [out]IEnumSession2 ** ppEnumSession);

        HRESULT EnumConnections(
                [in]WORD  TSIH,
                [out]IEnumConnection ** ppEnumConnection);

        HRESULT GetSession(
                [in]WORD  TSIH,
                [out]SESSION2 * pSessionInfo);

        HRESULT GetConnection(
                [in]WORD  TSIH,
                [in]DWORD  CID,
                [out]CONNECTION * pConnectionInfo);
};
```

7 Appendix B: Product Behavior

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

- Windows Server 2008 R2 operating system

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

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 2.2: The Windows implementation of the iSCSI Software Target does not use square-bracket delimiting or zero compression for the string representations of IPv6 addresses, although leading-zero suppression is allowed. An example of the IPv6 address form that the iSCSI Software Target expects is "21DA:D3:0:2F3B:2AA:FF:FE28:9C5A".

Additionally, the Windows implementation does not expect prefix designators (for example, "x:x:x:x:x:x:x:x/48") or scope identifiers (for example, "x:x:x:x:x:x:x:x%3"). Finally, the Windows implementation expects only global **unicast** and **multicast** IPv6 addresses, as described in [IANAIPV6AS].

<2> Section 2.2.3.9: The Windows implementation of the iSCSI Software Target does not support the **DynamicVhd** type.

<3> Section 2.2.4.4: In Windows the path is in **Universal Naming Convention (UNC)** form; that is, "\\.\system_disk_name", where "." in the place of <servername> in the UNC path stands for the local machine. For example, "\\.\PhysicalDisk0".

<4> Section 2.2.4.5: In Windows, the path is in UNC form; that is, "\\.\system_disk_name", where "." in the place of <servername> in the UNC path stands for the local machine. For example, "\\.\PhysicalDisk0".

<5> Section 2.2.4.8: Windows implementations do not use the **LICENSE** structure.

<6> Section 2.2.4.13: The Windows implementation does not use the **bSecure** field, but always sets it to false.

<7> Section 3.1.4.1.3: The EnumDisks method is implemented in Windows, but the resulting enumeration dataset is empty.

<8> Section 3.1.4.1.3: The EnumDisks2 method is implemented in Windows, but the resulting enumeration dataset is empty.

<9> Section 3.1.4.2.1: Windows implementations check whether the ISTM server target service is installed, configured, and running, and not in a transient startup or shutdown state.

<10> Section 3.1.4.2.1: Windows implementations check whether the ISTM server cluster service is installed, configured, and running, and not in a transient startup or shutdown state.

<11> Section 3.1.4.2.1: Windows implementations check whether the calling client belongs to the local Administrators group.

<12> Section 3.1.4.2.2.1: In Windows, the Next method returns a maximum of one item per call.

<13> Section 3.1.4.2.2.1: In Windows, the Next method performs parameter validation as follows:

If the *ulNumElements* parameter is nonzero, and the *pElements* parameter is NULL, E_POINTER is returned.

If the *ulNumElements* parameter is not equal to 1, and the *pNumElementsReturned* parameter is NULL, E_POINTER is returned.

If the *ulNumElements* parameter is greater than 1, and the *pNumElementsReturned* parameter is not NULL, E_FAIL is returned.

<14> Section 3.1.4.2.2.3: In Windows, the Reset method always returns S_OK.

<15> Section 3.3.4.1: In Windows, the **GetDriveString** method does not perform the specified function. It always returns E_NOTIMPL.

<16> Section 3.3.4.2: In the Windows implementation, when the **FindFirst** method is called, the ISTM server determines if an enumeration has already been started but not closed properly. If that is the case, the server closes the current enumeration.

<17> Section 3.6: Windows implementations do not use the **IEnumDisk** interface.

<18> Section 3.7: Windows implementations do not use the **IEnumDisk2** interface.

<19> Section 3.8.4.1: The Windows implementation of IHost supports IHost3.

<20> Section 3.21.4.11: In Windows, the **[propput]Status** method does not perform the specified function. It always returns E_NOTIMPL.

<21> Section 3.21.4.13: In Windows, the **[propput]LastLogIn** method does not perform the specified function. It always returns E_NOTIMPL.

<22> Section 3.21.4.26: The Windows implementation sets the default value for **TargetMaxRecvDataSegmentLength** to 65536.

<23> Section 3.21.4.27: The Windows implementation checks whether the value is between 512 and 16777215.

<24> Section 3.21.4.28: The Windows implementation sets the default value for **TargetFirstBurstLength** to 65536.

<25> Section 3.21.4.29: The Windows implementation checks whether the value is between 512 and 16777215.

<26> Section 3.21.4.30: The Windows implementation sets the default value of **TargetMaxBurstLength** to 262144.

<27> Section 3.21.4.31: The Windows implementation checks whether the value is between 512 and 16777215.

<28> Section 3.21.4.32: The Windows implementation sets the default value for **NumRecvBuffers** to 10.

<29> Section 3.21.4.33: The Windows implementation checks whether the number is between 1 and 128.

<30> Section 3.21.4.34: The Windows implementation sets the default value for **EnforceIdleTimeoutDetection** to true.

<31> Section 3.21.4.36: In Windows, the **[propget]ListenAllNetworkAdapters** method does not perform the specified function. It always returns E_NOTIMPL.

<32> Section 3.21.4.37: In Windows, the **[propput]ListenAllNetworkAdapters** method does not perform the specified function. It always returns E_NOTIMPL.

<33> Section 3.21.4.38: In Windows, the **[propget]Processor** method does not perform the specified function. It always returns E_NOTIMPL.

<34> Section 3.21.4.39: In Windows, the **[propput]Processor** method does not perform the specified function. It always returns E_NOTIMPL.

<35> Section 3.21.4.45: In the Windows implementation, the maximum number of LUNs allowed per target is 128.

<36> Section 3.21.4.48: In Windows, the maximum number of LUNs allowed per target is 128.

<37> Section 3.21.4.49: In the Windows implementation, the collection of portals servicing requests for the target is always empty.

<38> Section 3.21.4.50: In Windows, the **AddPortal** method does not perform the specified function. It always returns E_NOTIMPL.

<39> Section 3.21.4.51: In Windows, the **RemovePortal** method does not perform the specified function. It always returns S_OK.

<40> Section 3.21.4.52: In Windows, the **RemoveAllPortals** method does not perform the specified function. It always returns S_OK.

<41> Section 3.21.4.53: In Windows, the **SetCHAPInfoByBlob** method does not perform the specified function. It always returns E_NOTIMPL.

<42> Section 3.21.4.55: In Windows, the validation policy for the secret includes enforcement of a reasonable length, between 12 and 16 characters, and non-duplication with other secrets already established for the target.

<43> Section 3.21.4.56: In Windows, the validation policy for the secret includes enforcement of a reasonable length, between 12 and 16 characters, and non-duplication with other secrets already established for the target.

<44> Section 3.21.4.57: In Windows, the **CheckTargetRedirect** method does not perform the specified function. It always returns S_FALSE.

<45> Section 3.21.4.57: In the Windows implementation, the value of the *pRedirectAddr* parameter is undefined.

<46> Section 3.22.4.1: The Windows implementation of the ISTM client displays a message that shows the maximum supported limit of virtual disks per target.

<47> Section 3.23.4.4: In Windows, the **[propput]ResourceGroupName** method does not perform the specified function. It always returns E_NOTIMPL.

<48> Section 3.23.4.5: In Windows, the specified function is not performed when the **SaveToStream** method is called remotely.

<49> Section 3.23.4.5: In Windows, when the **SaveToStream** method is called locally, the server writes the target properties described in the target management abstract data model (section 3.1.1.7) to the instance of **IStream** [MSDN-IStream] that is derivable from the **IUnknown** interface specified by the *pStream* pointer.

<50> Section 3.24.4.1: In Windows, the resource state is the value returned by cluster resource API method **GetClusterResourceState()**.

<51> Section 3.24.4.2: In Windows, the **[propput]ResourceState** method does not perform the specified function. It always returns E_NOTIMPL.

<52> Section 3.26.4.1: The Windows client implementation displays a message that suggests the user choose a different name for the new target.

<53> Section 3.26.4.1: The Windows client implementation displays a message that suggests the user choose a shorter name for the new target.

<54> Section 3.27.4.3: In Windows, the specified function is not performed when the **OpenHost** method is called remotely.

<55> Section 3.27.4.3: In Windows, when the **OpenHost** method is called locally, the server performs the following processing:

Creates an **IHost** object and initializes it with the properties described in the target management abstract data model (section 3.1.1.7) that are loaded from the instance of **IStream** [MSDN-IStream] that is derivable from the **IUnknown** interface specified by the *pStream* pointer.

Brings the new target online.

Adds the **IHost** object to the collection of known targets.

Registers the target with the iSNS manager.

Returns the pointer to the **IHost** interface for the new target by using the *ppHost* parameter.

<56> Section 3.28.4.1: The Windows client implementation displays a message that suggests the user choose a different name for the new target.

<57> Section 3.28.4.1: The Windows client implementation displays a message that suggests the user choose a shorter name for the new target.

<58> Section 3.30.4.1: The Windows implementation of the **EnumDisks** method returns an empty enumeration.

<59> Section 3.30.4.3: In Windows, the **CreateVolume** method does not perform the specified function. It always returns E_NOTIMPL.

<60> Section 3.30.4.5: The Windows implementation always returns true.

<61> Section 3.30.4.6: In the Windows implementation, if the value in the *hyFileSize* parameter is less than 1 MB, the server returns E_INVALIDARG.

<62> Section 3.30.4.6: In the Windows implementation, if the value in the *hyFileSize* parameter is not on a MB boundary, the server truncates the value to the next lower value that is on a MB

boundary. If the resulting value is less than 8 MB or greater than the maximum file size for the operating system, the server returns ERROR_BAD_LENGTH.

<63> Section 3.30.4.7: In the Windows implementation, if the value in the *hyAdditionalSize* parameter is less than 1 MB, the server returns E_INVALIDARG.

If the value in the *hyAdditionalSize* parameter is not on a MB boundary, the server truncates the value to the next lower value that is on a MB boundary.

<64> Section 3.30.4.18: In Windows, the **GetProcessors** method does not perform the specified function. It always returns E_NOTIMPL.

<65> Section 3.31.4.1: The Windows implementation of the ISTM client displays a message indicating that the path is invalid because only an existing virtual disk that resides on a local hard disk of the iSCSI Software Target server is allowed to be specified.

<66> Section 3.31.4.1: The Windows implementation of the ISTM client displays a message indicating that the VHD file does not exist or is not a valid file.

<67> Section 3.31.4.1: The Windows implementation of the ISTM client displays a message indicating that the format of the VHD file is not supported.

<68> Section 3.31.4.2: The Windows implementation of the ISTM client displays a message indicating that the virtual disk cannot be created because the virtual disk size is required to be greater than 8 MB and less than 2048 GB (2 terabytes).

<69> Section 3.31.4.3: The Windows implementation of the ISTM client displays a message indicating that the specified path is already in use and that the user needs to specify a different path.

<70> Section 3.31.4.3: The Windows implementation of the ISTM client displays a message indicating that a virtual disk can be created only on a local hard disk of the iSCSI Software Target computer, not on another virtual disk.

<71> Section 3.32.4.1: The Windows implementation of the **EnumDisks2** method returns an empty enumeration.

<72> Section 3.33.4.1: In the Windows implementation, the maximum file size is 2 terabytes (2,199,023,255,552 bytes).

<73> Section 3.36.4.2: In Windows, the **[propput]Id** method does not perform the specified function. It always returns E_NOTIMPL.

<74> Section 3.36.4.3: In Windows, the **[propput]VssSnapshotId** method does not perform the specified function. It always returns E_NOTIMPL.

<75> Section 3.36.4.4: In Windows, the **[propput]VssSnapshotSetId** method does not perform the specified function. It always returns E_NOTIMPL.

<76> Section 3.36.4.6: In Windows, the **[propput]OrigWTD** method does not perform the specified function. It always returns E_NOTIMPL.

<77> Section 3.36.4.9: In Windows, the **[propput]ExportedWTD** method does not perform the specified function. It always returns E_NOTIMPL.

<78> Section 3.36.4.12: In Windows, the **[propput]Status** method does not perform the specified function. It always returns E_NOTIMPL.

<79> Section 3.37.4.1: The Windows implementation of the ISTM client displays a message that describes possible causes of the rollback failure, such as the virtual disk is in use by an iSCSI initiator, or it is locally mounted, and possible remedies, such as disconnecting initiators or unmounting the virtual disk.

<80> Section 3.38.4.3: The Windows implementation of the ISTM server does not support snapshots of LUNs of type **DifferencingVhd**.

<81> Section 3.38.4.7: In Windows, the **ImportSnapshot** method does not perform the specified function. It always returns E_NOTIMPL.

<82> Section 3.39.4.1: The Windows implementation of the ISTM client displays a message that describes possible causes of the failure to delete the snapshot, such as the snapshot might be in use in a rollback operation or the virtual disk associated with the snapshot might be locally mounted.

<83> Section 3.39.4.1: The Windows implementation of the ISTM client displays a message that describes possible causes of the failure to delete the snapshot, such as the snapshot is in use, and possible remedies, such as disconnecting all iSCSI initiators that are using the virtual disk.

<84> Section 3.40.3: The Windows implementation obtains this information from the **registry**.

<85> Section 3.40.3: The Windows implementation does not maintain a persisted collection of cached iSCSI initiators. When an ISTM client makes a request for a cached initiator, that information is obtained by querying the iSNS servers.

<86> Section 3.40.4.2: The Windows implementation adds the specified iSNS server to the registry if it was not discovered using the Dynamic Host Configuration Protocol (DHCP).

<87> Section 3.40.4.3: The Windows implementation does not remove the specified iSNS server if it was discovered using the Dynamic Host Configuration Protocol (DHCP). Otherwise, the Windows implementation also removes the specified iSNS server from the registry.

<88> Section 3.40.4.4: The Windows implementation does not maintain a persisted collection of cached iSCSI initiators. When an ISTM client makes a request for a cached initiator, that information is obtained by querying the iSNS servers.

<89> Section 3.43.4.4: In Windows, the **[propput]WTD** method does not perform the specified function. It always returns E_NOTIMPL.

<90> Section 3.43.4.6: In Windows, the **[propput]Type** method does not perform the specified function. It always returns E_NOTIMPL.

<91> Section 3.43.4.8: In Windows, the **[propput]Flags** method does not perform the specified function. It always returns E_NOTIMPL.

<92> Section 3.43.4.10: In Windows, the **[propput]Status** method does not perform the specified function. It always returns E_NOTIMPL.

<93> Section 3.43.4.11: In Windows, the **[propget]Assigned** method does not perform the specified function. It always returns E_NOTIMPL.

<94> Section 3.43.4.12: In Windows, the **[propput]Assigned** method does not perform the specified function. It always returns E_NOTIMPL.

<95> Section 3.43.4.21: In the Windows implementation, the minimum value for a snapshot store is 320 MB. A value of -1 in the *nSize* parameter indicates that the limit for the size of the snapshot store depends only on available resources.

<96> Section 3.43.4.24: In Windows, the **[propput]LMType** method does not perform the specified function. It always returns E_NOTIMPL.

<97> Section 3.43.4.27: In Windows, the **[propget]LMWTManageMountPoint** method does not perform the specified function. It always returns E_NOTIMPL.

<98> Section 3.43.4.28: In Windows, the **[propput]LMWTManageMountPoint** method does not perform the specified function. It always returns E_NOTIMPL.

<99> Section 3.43.4.29: In Windows, the **[propget]LMMountPoint** method does not perform the specified function. It always returns E_NOTIMPL.

<100> Section 3.43.4.30: In Windows, the [**propput]LMMountPoint** method does not perform the specified function. It always returns E_NOTIMPL.

<101> Section 3.43.4.32: In Windows, the **[propput]LMCurrMountDeviceId** method does not perform the specified function. It always returns E_NOTIMPL.

<102> Section 3.44.4.2: Windows supports only fixed and differencing VHD type virtual disks.

<103> Section 3.44.4.6: In Windows, the **[propput]ResourceGroupName** method does not perform the specified function. It always returns E_NOTIMPL.

<104> Section 3.44.4.7: In Windows, the **IsAlive** method does not perform the specified function. It always returns S_OK.

<105> Section 3.44.4.8: In Windows, the specified function is not performed when the **SaveToStream** method is called remotely.

<106> Section 3.44.4.8: In Windows, when the **SaveToStream** method is called locally, the server writes the virtual disk settings described in the LUN management abstract data model (section 3.1.1.4) to the instance of **IStream** [MSDN-IStream] that is derivable from the **IUnknown** interface specified by the *pStream* pointer.

<107> Section 3.45.4.7: In Windows, the **[propput]ResourceState** method does not perform the specified function. It always returns E_NOTIMPL.

<108> Section 3.46.4.2: Windows supports only **WTDTypeFileBased** and **WTDTypeRamBased** device types.

<109> Section 3.46.4.3: In Windows, the **CreateShadowWTDisk** method does not perform the specified function.

<110> Section 3.46.4.3: Windows supports only **WTDTypeFileBased** and **WTDTypeRamBased** device types.

<111> Section 3.46.4.6: In Windows, the **EnumWTDisksOnSameVolumeByPath** method does not perform the specified function. It always returns E_NOTIMPL.

<112> Section 3.46.4.8: In Windows, the **DVMountSelectedWTDisksSnapshotNow** method does not perform the specified function. It always returns S_OK.

<113> Section 3.46.4.9: In Windows, the **ImportWTDisk** method does not perform the specified function. It always returns E_NOTIMPL.

<114> Section 3.47.4.1: The Windows implementation of the ISTM client displays a message that suggests the user choose a different name for the virtual disk to be created.

<115> Section 3.47.4.1: The Windows implementation of the ISTM client displays a message that defines the valid range of disk size, which is greater than 8 megabytes and less than 2 terabytes.

<116> Section 3.47.4.2: The Windows implementation of the ISTM client displays a message that describes a possible cause of the failure to delete the virtual disk, such as the virtual disk is in use, and a possible remedy, such as disconnecting all iSCSI initiators that are using the disk.

<117> Section 3.47.4.2: The Windows implementation of the ISTM client displays a message that describes a possible cause of the failure to delete the virtual disk, such as the virtual disk or a snapshot of the virtual disk might be locally mounted. The message also suggests dismounting the device and retrying the operation.

<118> Section 3.47.4.2: The Windows implementation of the ISTM client displays a message that describes a possible cause of the failure to delete the virtual disk, such as the virtual disk has a dependency, such as snapshots with exported virtual disks.

<119> Section 3.48.4.1: Windows supports only **WTDTypeFileBased** and **WTDTypeRamBased** device types.

<120> Section 3.48.4.2: Windows supports only **WTDTypeFileBased** and **WTDTypeRamBased** device types.

<121> Section 3.48.4.3: In Windows, the specified function is not performed when the **OpenWTDisk** method is called remotely.

<122> Section 3.48.4.3: In Windows, when the **OpenWTDisk** method is called locally, the server performs the following processing:

Creates an **IWTDisk** object and initializes it with the properties described in the LUN management abstract data model (section 3.1.1.4) that are loaded from the instance of **IStream** [MSDN-IStream] that is derivable from the **IUnknown** interface specified by the *pStream* pointer.

Brings the new virtual disk online.

Adds the **IWTDisk** object to the collection of known virtual disks.

Sends notifications of the change in the LUN list to registered clients.

Returns the pointer to the **IWTDisk** interface for the new virtual disk by using the *ppWTDisk* parameter.

<123> Section 3.50.4.1: In Windows, the **ClusterNotifyWTDiskDeletion** method does not perform the specified function. It always returns E_NOTIMPL.

<124> Section 3.52.4.2: In the Windows implementation, the computer name is returned by default.

<125> Section 3.52.4.4: In Windows, the **[propget]License** method does not perform the specified function. It always returns E_NOTIMPL.

<126> Section 3.52.4.5: The Windows implementation of the **[propget]WTCliPath** method returns the path to a command-line executable application that is used for managing snapshot backup schedules.

<127> Section 3.52.4.7: In Windows, the **AddLicense** method does not perform the specified function. It always returns E_NOTIMPL.

<128> Section 3.52.4.11: In Windows, the **DVGetSnapshotSchedule** method does not perform the specified function. It always returns E_NOTIMPL.

<129> Section 3.53.4.1: The Windows client implementation displays a message that instructs the end user in performing basic checks to remove the possible causes of unavailability of the iSCSI Software Target service.

<130> Section 3.53.4.1: The Windows client implementation displays a message that informs the end user that membership is required in the Administrators group on the iSCSI Software Target computer.

<131> Section 3.53.4.2: The Windows client implementation displays a message that instructs the end user in performing basic checks to remove the possible causes of unavailability of the iSCSI Software Target service.

<132> Section 3.53.4.2: The Windows client implementation displays a message that informs the end user that membership is required in the Administrators group on the iSCSI Software Target computer.

8 Change Tracking

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

9 Index