[MS-DLTM-Diff]:

Distributed Link Tracking: Central Manager Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

Tools. The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
3/2/2007	1.0	New	Version 1.0 release
4/3/2007	1.1	Minor	Version 1.1 release
5/11/2007	1.2	Minor	Version 1.2 release
6/1/2007	1.2.1	Editorial	Changed language and formatting in the technical content.
7/3/2007	2.0	Major	Converted to unified format.
8/10/2007	3.0	Major	Updated and revised the technical content.
9/28/2007	4.0	Major	Updated and revised the technical content.
10/23/2007	4.1	Minor	Updated the IDL.
1/25/2008	4.1.1	Editorial	Changed language and formatting in the technical content.
3/14/2008	4.1.2	Editorial	Changed language and formatting in the technical content.
6/20/2008	5.0	Major	Updated and revised the technical content.
7/25/2008	5.0.1	Editorial	Changed language and formatting in the technical content.
8/29/2008	6.0	Major	Updated and revised the technical content.
10/24/2008	6.0.2	Editorial	Changed language and formatting in the technical content.
12/5/2008	6.0.3	Editorial	Changed language and formatting in the technical content.
1/16/2009	6.0.4	Editorial	Changed language and formatting in the technical content.
2/27/2009	6.0.5	Editorial	Changed language and formatting in the technical content.
4/10/2009	6.0.6	Editorial	Changed language and formatting in the technical content.
5/22/2009	6.0.7	Editorial	Changed language and formatting in the technical content.
7/2/2009	6.0.8	Editorial	Changed language and formatting in the technical content.
8/14/2009	6.1	Minor	Clarified the meaning of the technical content.
9/25/2009	7.0	Major	Updated and revised the technical content.
11/6/2009	7.0.1	Editorial	Changed language and formatting in the technical content.
12/18/2009	7.1	Minor	Clarified the meaning of the technical content.
1/29/2010	7.1.1	Editorial	Changed language and formatting in the technical content.
3/12/2010	7.1.2	Editorial	Changed language and formatting in the technical content.
4/23/2010	7.1.3	Editorial	Changed language and formatting in the technical content.
6/4/2010	7.1.4	Editorial	Changed language and formatting in the technical content.
7/16/2010	8.0	Major	Updated and revised the technical content.
8/27/2010	9.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
10/8/2010	9.0	None	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	9.0	None	No changes to the meaning, language, or formatting of the technical content.
1/7/2011	9.0	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2011	9.0	None	No changes to the meaning, language, or formatting of the technical content.
3/25/2011	9.0	None	No changes to the meaning, language, or formatting of the technical content.
5/6/2011	9.0	None	No changes to the meaning, language, or formatting of the technical content.
6/17/2011	9.1	Minor	Clarified the meaning of the technical content.
9/23/2011	9.1	None	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	9.1	None	No changes to the meaning, language, or formatting of the technical content.
3/30/2012	9.1	None	No changes to the meaning, language, or formatting of the technical content.
7/12/2012	9.1	None	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	9.1	None	No changes to the meaning, language, or formatting of the technical content.
1/31/2013	9.1	None	No changes to the meaning, language, or formatting of the technical content.
8/8/2013	9.1	None	No changes to the meaning, language, or formatting of the technical content.
11/14/2013	9.1	None	No changes to the meaning, language, or formatting of the technical content.
2/13/2014	9.1	None	No changes to the meaning, language, or formatting of the technical content.
5/15/2014	9.1	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	9.1	None	No changes to the meaning, language, or formatting of the technical content.
10/16/2015	9.1	None	No changes to the meaning, language, or formatting of the technical content.
7/14/2016	9.1	None	No changes to the meaning, language, or formatting of the technical content.
6/1/2017	9.1	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the Distributed Link Tracking: Central Manager Protocol.

Distributed Link Tracking (DLT) consists of two protocols that work together to discover the new location of a file that has moved. DLT can determine whether the file has moved on a mass-storage device, within a computer, or between computers in a network. In addition to the Distributed Link Tracking: Central Manager Protocol, DLT includes the Distributed Link Tracking: Workstation Protocol, as specified in [MS-DLTW], which is used to determine a file's current location. Both DLT protocols are remote procedure call (RPC) interfaces.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

authenticated user: A built-in security group specified in [MS-WSO] whose members include all users that can be authenticated by a computer.

ClientVolumeTable: This table has an entry for each of the volumes of the computer on which it runs.

Distributed Link Tracking (DLT): A protocol that enables client applications to track sources that have been sent to remote locations using remote procedure call (RPC) interfaces, and to maintain links to files. It exposes methods that belong to two interfaces, one of which exists on the server (trksvr) and the other on a workstation (trkwks).

DLT: See Distributed Link Tracking (DLT).

domain: A set of users and computers sharing a common namespace and management infrastructure. At least one computer member of the set must act as a domain controller (DC) and host a member list that identifies all members of the domain, as well as optionally hosting the Active Directory service. The domain controller provides authentication ~~(2)~~ of members, creating a unit of trust for its members. Each domain has an identifier that is shared among its members. For more information, see [MS-AUTHSOD] section 1.1.1.5 and [MS-ADTS].

domain controller (DC): The service, running on a server, that implements Active Directory, or the server hosting this service. The service hosts the data store for objects and interoperates with other DCs to ensure that a local change to an object replicates correctly across all DCs. When Active Directory is operating as Active Directory Domain Services (AD DS), the DC contains full NC replicas of the configuration naming context (config NC), schema naming context (schema NC), and one of the domain NCs in its forest. If the AD DS DC is a global catalog server (GC server), it contains partial NC replicas of the remaining domain NCs in its forest. For more information, see [MS-AUTHSOD] section 1.1.1.5.2 and [MS-ADTS]. When Active Directory is operating as Active Directory Lightweight Directory Services (AD LDS), several AD LDS DCs can run on one server. When Active Directory is operating as AD DS, only one AD DS DC can run on one server. However, several AD LDS DCs can coexist with one AD DS DC on one server. The AD LDS DC contains full NC replicas of the config NC and the schema NC in its forest. The domain controller is the server side of Authentication Protocol Domain Support [MS-APDS].

domain controller locator: A function within a domain that provides for location of a domain controller (DC) and the ability to determine certain properties of DCs. For more information, see [MS-ADTS].

dynamic endpoint: A network-specific server address that is requested and assigned at run time. For more information, see [C706].

FileId: The FileLocation of a file at the time it was originally created. A file's FileId never changes.

FileInformation: Information that is maintained about a file, such as its FileId and/or ObjectID.

FileLinkInformation: Information about a file that is necessary to identify and locate that file, including the file's last known Universal Naming Convention (UNC) name, the MachineID of the computer on which the file was last known to be located, the last known FileLocation of the file, and the file's permanent FileID.

FileLocation: A VolumeID with an appended ObjectID, which together represent the location of a file at some point in time, though the file might no longer be there. FileLocation values are stored in droid (CDomainRelativeObjId) data structures.

FileTable: A table (with rows uniquely identified by a FileLocation or FileID) that contains the following fields: [PreviousFileLocation, FileLocation, FileID, RefreshTime]. For more information [MS-DLTM] see section 3.1.1. Maps a FileLocation or FileID to a current FileLocation.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the GUID. See also universally unique identifier (UUID).

Interface Definition Language (IDL): The International Standards Organization (ISO) standard language for specifying the interface for remote procedure calls. For more information, see [C706] section 4.

MachineID: A unique identifier that represents the identity of a computer.

MoveNotificationCursor: A value that indicates the next entry in the MoveNotificationList that is to be sent to the server in a MOVE_NOTIFICATION message. See section 3.2.1 for more information.

MoveNotificationList: A list of records with information about files that have been moved from a volume on this machine. See section 3.2.1 for more information.

MoveNotificationVolumeCursor: A value that indicates that a sequence of MOVE_NOTIFICATION messages is in progress, and indicates the current volume (from the ClientVolumeTable) for which notifications are being sent. See section 3.2.5.6 for more information.

ObjectID: A unique identifier that represents the identity of a file within a file system volume. For more information, see [MS-DLTM].

PreviousFileLocation: The FileLocation of a file before it was moved.

principal name: The computer or user name that is maintained and authenticated by the Active Directory directory service.

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

RequestMachine: The MachineID of the computer that is the client calling the LnkSvrMessage method.

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

security provider: A pluggable security module that is specified by the protocol layer above the remote procedure call (RPC) layer, and will cause the RPC layer to use this module to secure messages in a communication session with the server. The security provider is sometimes referred to as an authentication service. For more information, see [C706] and [MS-RPCE].

ServerVolumeTable: A table (with rows uniquely identified by a VolumeID) that contains the following fields: [VolumeID, VolumeSequenceNumber, VolumeSecret, VolumeOwner, RefreshTime]. For more information see section 3.1.1.

Subrequest: A request within a SYNC_VOLUMES request. For details on requests, see section 3.1.4.

Universal Naming Convention (UNC): A string format that specifies the location of a resource. For more information, see [MS-DTYP] section 2.2.57.

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

user principal name (UPN): A user account name (sometimes referred to as the user logon name) and a domain name that identifies the domain in which the user account is located. This is the standard usage for logging on to a Windows domain. The format is: someone@example.com (in the form of an email address). In Active Directory, the userPrincipalName attribute ~~(2)~~ of the account object, as described in [MS-ADTS].

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

VolumeFileTable: A table that contains FileInformation entries, and is maintained for each volume in the ClientVolumeTable. Like the ClientVolumeTable, this table MUST be stored on the volume itself. The VolumeFileTable has an entry for each file on the volume that is to be tracked with this protocol. Each entry has a CrossVolumeMoveFlag value, which MUST initially be set to zero. For more information see section 3.2.1.

VolumeID: A unique identifier that represents the identity of a file system volume.

VolumeInformation: A table that maintains volume information for each volume in the ClientVolumeTable.

VolumeOwner: A MachineID that is considered to be the owner of a VolumeID. A VolumeID can only have one VolumeOwner. For more information, see [MS-DLTM].

VolumeSecret: A value that is used to establish a VolumeOwner. For more information, see [MS-DLTM].

VolumeSequenceNumber: An integer value used to track the sequence of move notification messages received by the protocol server. See [MS-DLTM] section 2.2.2 for more information.

workstation: A terminal or desktop computer in a network that is used to run applications and is connected to a server from which it obtains data shared with other computers.

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

1.2 References

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

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

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

[MS-ADTS] Microsoft Corporation, "Active Directory Technical Specification".

[MS-DLTW] Microsoft Corporation, "Distributed Link Tracking: Workstation Protocol".

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

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

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

[MS-SAMR] Microsoft Corporation, "Security Account Manager (SAM) Remote Protocol (Client-to-Server)".

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol".

[MS-SPNG] Microsoft Corporation, "Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) Extension".

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

1.2.2 Informative References

[MS-FSCC] Microsoft Corporation, "File System Control Codes".

[MSDN-FNDFSTVOL] Microsoft Corporation, "FindFirstVolume function", http://msdn.microsoft.com/en-us/library/aa364425.aspx

[MSDN-FNDNXTVOL] Microsoft Corporation, "FindNextVolume function", http://msdn.microsoft.com/en-us/library/aa364431.aspx

[MSDN-GETDRIVETYPE] Microsoft Corporation, "GetDriveType function", http://msdn.microsoft.com/en-us/library/aa364939.aspx

[MSDN-GETVOLINFO] Microsoft Corporation, "GetVolumeInformation function", http://msdn.microsoft.com/en-us/library/Aa364993.aspx

[MSDN-SHELLLINKS] Microsoft Corporation, "Shell Links", http://msdn.microsoft.com/en-us/library/bb776891.aspx

1.3 Overview

The Distributed Link Tracking: Central Manager Protocol is based on the Remote Procedure Call Protocol Extensions, as specified in [MS-RPCE]. The primary purpose of this protocol is to allow clients of the Distributed Link Tracking: Workstation Protocol to determine the correct server to contact when searching for a file. To accomplish this, the Distributed Link Tracking (DLT) Central Manager server accepts notifications of file and volume moves, in addition to other relevant information from participating computers. This information can be queried by clients to get the file's current location in UNC form.

The following is a scenario of this protocol working together with the Distributed Link Tracking: Workstation Protocol:

- A file is created on computer M1. M1 assigns identifiers, specifically FileID and FileLocation, to the file.
- Computer M0 makes note of the file, locally storing its identifiers.
- The file is moved from computer M1 to M2 and from there to M3. In conjunction with these moves, the file maintains its FileID value, but a new FileLocation identifier is assigned.
- To find the file in its new location, M0 contacts a DLT Central Manager server to query the current location of the file.
- The DLT Central Manager server queries its tables, and determines that the file is currently on computer M3.
- M0 contacts the Distributed Link Tracking: Central Manager Protocol on M3, and learns the file's new name and location.

The following diagram shows the machine configuration for this example. The list after the diagram walks through the scenario, and describes in more detail how the Distributed Link Tracking: Central Manager Protocol is used.

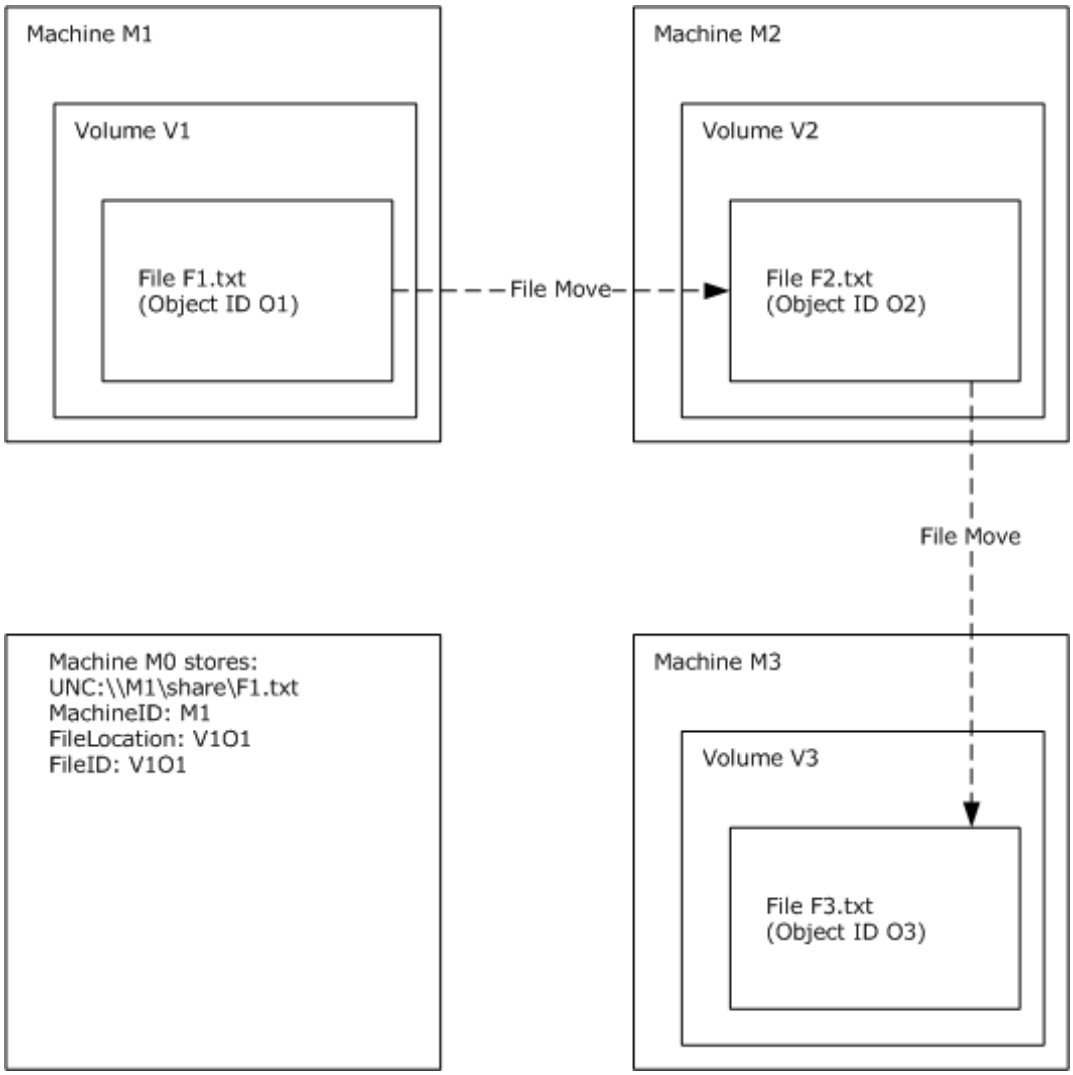

Figure 1: Distributed Link Tracking (DLT) process

- A file named F1.txt starts on a computer with MachineID M1, on a volume with VolumeID V1, and with an ObjectID of O1.
- The file is first moved to a computer with MachineID M2. On M2, the file has a new name, F2.txt, and a new ObjectID of O2. The volume on that computer has a VolumeID of V2.
- M1 sends a notification to the DLT Central Manager server indicating that the file with FileLocation V1O1 and FileID V1O1 has been moved to M2, where it now has the FileLocation V2O2.
- The file is next moved to computer M3. On that computer, the file has the name F3.txt, and new ObjectID O3. The volume on that computer has a VolumeID of V3.
- M2 sends a notification to the DLT Central Manager server indicating that the file with FileLocation V2O2 and FileID V1O1 has been moved to M3, where it now has the FileLocation V3O3. Note that this notification is sent to the same DLT Central Manager server instance as the notification from M1.

- Before the file is moved, a user on computer M0 requests that information about the file be saved, so that its location can be determined after it has been moved.<1> As a result, M0 stores the UNC, the MachineID, the FileLocation, and the FileID.
- After the file has been moved both times, and after M0 attempts to open the file, the DLT Workstation client on M0 sends a query to the DLT Workstation server on M1, requesting the new UNC for the file with FileID V1O1.<2> That server responds that the file has been moved to M2, with a new FileLocation of V2O2.
- Rather than contacting M2, M0 then sends a request to a DLT Central Manager server. Note that this is the same server that received the move notifications from M1 and M2, or a server instance that shares tables with the servers that received those notifications.
- The DLT Central Manager server returns the file's current information, according to its tables, that the file is now on computer M3, with FileLocation V3O3.
- M0 then makes a request to the DLT Workstation server on M3, which indicates that the file now has the UNC "\\M3\share\F3.txt".
- M0 updates its store of information for the file with these updated UNC and FileLocation values.<3>

1.4 Relationship to Other Protocols

The Distributed Link Tracking: Central Manager Protocol is dependent upon RPC, as specified in [C706] and [MS-RPCE], which is its transport protocol.

This protocol is designed to support clients of the Distributed Link Tracking: Workstation Protocol. Clients of the Distributed Link Tracking: Workstation Protocol use the Distributed Link Tracking: Central Manager Protocol to determine the correct computer on which to send a DLT Workstation server request.

DLT Central Manager servers run only on domain controller (DC) computers.

1.5 Prerequisites/Preconditions

The Distributed Link Tracking: Central Manager Protocol is composed of RPC interfaces, and as a result, has the prerequisites identified in [C706] as those common to RPC interfaces.

The security configuration for using RPC is specified further in section 2.1, section 3.1.3, and section 3.2.3.

1.6 Applicability Statement

The Distributed Link Tracking: Central Manager Protocol applies to computers in a domain that shares files with the Server Message Block (SMB) Protocol, as specified in [MS-SMB], when each of the following is true:

- The computers hold files.
- References are made to files.
- Files are moved between computers or within a computer, or the computer name changes.
- File references are meant to be found after the referent file has been moved in one of these ways, either between computers or within a computer.

This protocol is not intended for use in situations where large numbers of files are to be tracked. In such situations, limits in the protocol are reached. These limits are specified in section 3.1.4.4.4 for

the TRK_E_VOLUME_QUOTA_EXCEEDED return value, and in section 3.1.4.2 for the TRK_S_NOTIFICATION_QUOTA_EXCEEDED return value.

1.7 Versioning and Capability Negotiation

This protocol contains no capability negotiation issues. This document covers versioning issues in the following areas:

- Supported Transports: This protocol uses RPC over TCP/IP only, as specified in section 2.1.
- Protocol Versions: This protocol has only one interface version. The use of these methods is specified in section 3.1.4.
- Security and Authentication Methods: This protocol uses Simple and Protected Negotiation Mechanism (SPNEGO) and RPC packet authentication levels for security and authentication, as specified in section 2.1 and 3.2.3.
- Localization: The protocol does not contain locale-dependent information.

1.8 Vendor-Extensible Fields

The Distributed Link Tracking: Central Manager Protocol does not define any vendor-extensible fields, other than fields that contain GUIDs as specified in [MS-DTYP] section 2.3.4.2. The methods for generating such values are specified in [C706] Appendix A.

This protocol uses HRESULTs as defined in [MS-ERREF] and discussed in HRESULT (section 2.2.1). Vendors are free to choose their own values for this field, as long as the C bit (0x20000000) is set, indicating that it is a customer code.

1.9 Standards Assignments

The Distributed Link Tracking: Central Manager Protocol has no standards assignments. It uses the following UUID to identify its interfaces.

Parameter	Value	Reference
RPC interface UUID, as specified in [MS-DTYP] (section 2.3.4.1), for the Distributed Link Tracking: Central Manager Protocol	4da1c422-943d-11d1-acae-00c04fc2aa3f	[C706]; also see section 2.1 of this document.

2 Messages

2.1 Transport

The Distributed Link Tracking: Central Manager Protocol uses the RPC protocol sequence ncacn_ip_tcp, specified in [MS-RPCE] section 2.1.1.1.

This protocol uses RPC dynamic endpoints, as specified in [C706] section 6.2.2.

This protocol MUST use the following parameters:

- **UUID**: The RPC interface UUID value defined in [MS-DLTW] section 1.9 MUST be used.
- Version number: 1.0.
- A SPNEGO security provider. For more information on SPNEGO, see [MS-SPNG].
- A server principal name (see [MS-RPCE], section 3.2.2.3.2), calculated as follows:
  - ServerPrincipalName: DomainName + "\" + ServerMachineName + "$".
  - DomainName: Name of the domain in which the machine is a member.

2.2 Common Data Types

In addition to RPC base types, the following sections use the definition of GUID.

2.2.1 HRESULT

HRESULT, which is defined in [MS-ERREF], is a 32-bit signed integer returned by RPC method calls. Except where otherwise noted, a negative value indicates an error, and a non-negative value denotes success.

For example, in the Distributed Link Tracking: Central Manager Protocol, HRESULT is returned by the LnkSvrMessage method.

2.2.2 SequenceNumber

The SequenceNumber type stores a VolumeSequenceNumber value, which is used to track the sequence of move notification messages that are received by the DLT Central Manager server.

This type is declared as follows:

```
typedef signed long SequenceNumber;
```

When this value reaches its maximum positive value (2147483647), incrementing MUST result in its most negative value (-2147483648). For details, see sections 3.1.4.2 and 3.2.4.1.

2.2.3 CMachineId

The CMachineId type stores a MachineID value.

This type is defined in [MS-DLTW] section 2.2.2.

2.2.4 CDomainRelativeObjId

The CDomainRelativeObjId type stores a FileID or FileLocation. When used as a FileLocation, CDomainRelativeObjId represents a previous or current location of the file within the domain. When used as a FileID, CDomainRelativeObjId represents the file's original FileLocation, as well as the unique identifier for the file.

This type is composed of a CVolumeId type (representing the VolumeID) and a CObjId type (the identifier of a file relative to volume indicated by the VolumeID).

This type is defined in [MS-DLTW] section 2.2.3.

2.2.5 CVolumeId

The CVolumeId type stores the VolumeID of a volume.

This type is defined in [MS-DLTW] section 2.2.4.

2.2.6 CObjId

The CObjId type stores the ObjectID for a file. This is the unique identifier of a file within a volume.

This type is defined in [MS-DLTW] section 2.2.5.

2.2.7 CVolumeSecret

The CVolumeSecret type stores a VolumeSecret value, which is used to authenticate a VolumeOwner value. For an example, see section 3.1.4.4.1, which describes the processing of a CLAIM_VOLUME message.

```
typedef struct CVolumeSecret {
  byte _abSecret[8];
} CVolumeSecret;
```

_abSecret: An 8-byte volume password. The content of these bytes is arbitrary and is generated by the client. See section 3.2.5.3 for an example of where this value is used.

2.2.8 TRK_FILE_TRACKING_INFORMATION

The TRK_FILE_TRACKING_INFORMATION structure is used in a SEARCH message of a LnkSvrMessage method call to search for the current location of a file. This structure contains information about a file that is being tracked. See section 2.2.11 for more information about the SEARCH message. See section 3.2.6.3 for an example of how the TRK_FILE_TRACKING_INFORMATION structure is used.

```
typedef struct {
  CDomainRelativeObjId droidBirth;
  CDomainRelativeObjId droidLast;
  CMachineId mcidLast;
  HRESULT hr;
} TRK_FILE_TRACKING_INFORMATION;
```

droidBirth: The FileID of the file for which the location is being requested. For details on this structure, see [MS-DLTW] section 2.2.3.

droidLast: On input, the last FileLocation that the client knew of for the file. On output, this member contains the file's current FileLocation.

mcidLast: On completion of the SEARCH request, this member is returned by the server to indicate the MachineID of the VolumeOwner of the VolumeID component of the **droidLast** field. The CMachineId type is specified in [MS-DLTW] section 2.2.2.

hr: Return value that indicates the success or failure of this message. The type of this field is an HRESULT but, unlike the standard definition, only zero is a successful return value for this field. Any nonzero value MUST be treated identically as a failure value.

2.2.9 old_TRK_FILE_TRACKING_INFORMATION

The old_TRK_FILE_TRACKING_INFORMATION structure is unused, but is included in this protocol because it affects the definition of the TRKSVR_MESSAGE_UNION, as defined in section 2.2.13.

```
typedef struct {
  WCHAR tszFilePath[257];
  CDomainRelativeObjId droidBirth;
  CDomainRelativeObjId droidLast;
  HRESULT hr;
} old_TRK_FILE_TRACKING_INFORMATION;
```

2.2.10 TRKSVR_MESSAGE_PRIORITY

The TRKSVR_MESSAGE_PRIORITY enumeration contains constants that indicate the priority level of messages that can be sent by the LnkSvrMessage method. It is used in the TRKSVR_MESSAGE_UNION structure. See section 3.1.4.1 for more information on the LnkSvrMessage method.

```
typedef [v1_enum] enum
{
  PRI_0 = 0,
  PRI_1 = 1,
  PRI_2 = 2,
  PRI_3 = 3,
  PRI_4 = 4,
  PRI_5 = 5,
  PRI_6 = 6,
  PRI_7 = 7,
  PRI_8 = 8,
  PRI_9 = 9
} TRKSVR_MESSAGE_PRIORITY;
```

PRI_0: Priority 0, the highest priority.

PRI_1: Priority 1.

PRI_2: Priority 2.

PRI_3: Priority 3.

PRI_4: Priority 4.

PRI_5: Priority 5.

PRI_6: Priority 6.

PRI_7: Priority 7.

PRI_8: Priority 8.

PRI_9: Priority 9, the lowest priority.

2.2.11 TRKSVR_MESSAGE_TYPE

The TRKSVR_MESSAGE_TYPE enumeration defines the type of a message that is sent to the DLT Central Manager server by the LnkSvrMessage method. That is, the LnkSvrMessage method is defined such that the caller can send different messages in the form of different structures, and the TRKSVR_MESSAGE_TYPE enumeration is used in the TRKSVR_MESSAGE_UNION parameter of the LnkSvrMessage method to indicate which structure is being passed in a method call. See section 2.2.12 for information on TRKSVR_MESSAGE_UNION, and section 3.1.4.1 for more information on the LnkSvrMessage method.

```
typedef [v1_enum] enum
{
  old_SEARCH,
  MOVE_NOTIFICATION = 1,
  REFRESH = 2,
  SYNC_VOLUMES = 3,
  DELETE_NOTIFY = 4,
  STATISTICS = 5,
  SEARCH = 6,
  WKS_CONFIG,
  WKS_VOLUME_REFRESH
} TRKSVR_MESSAGE_TYPE;
```

old_SEARCH: Unused.

MOVE_NOTIFICATION: The message includes information about one or more files that have been moved (see section 3.2.6.1). The message data in this method call is formatted in the TRSVR_CALL_MOVE_NOTIFICATION structure (see section 2.2.12.1).

REFRESH: The message communicates the current status of the entries in the ClientVolumeTable, so that the server can update its ServerVolumeTable (see section 3.2.5.2). The message data in this method call is formatted in the TRKSVR_CALL_REFRESH structure (see section 2.2.12.2).

SYNC_VOLUMES: The message is used to synchronize a volume (see sections 3.2.5.3, 3.2.5.4, and 3.2.6.4). The message data in this method call is formatted in the TRKSVR_CALL_SYNC_VOLUMES structure (see section 2.2.12.3).

DELETE_NOTIFY: The message includes information about one or more files that have been deleted (see section 3.2.5.5). The message data in this method call is formatted in the TRKSVR_CALL_DELETE structure (see section 2.2.12.4).

STATISTICS: The message is a request for usage statistics. This message type is not used in this protocol, but is included in this specification because it affects the size of the TRKSVR_MESSAGE_UNION structure as it is transmitted over the RPC Protocol. See section 2.2.12 for more information on the TRKSVR_MESSAGE_UNION.

SEARCH: The message is a request for information about a moved file's updated location. (see section 3.2.6.3). The message data in this method call is formatted in the TRKSVRidl_struct_page_CALL_SEARCH structure (see section 2.2.12.6).

WKS_CONFIG: Unused.

WKS_VOLUME_REFRESH: Unused.

2.2.12 TRKSVR_MESSAGE_UNION

The TRKSVR_MESSAGE_UNION structure is used in LnkSvrMessage method calls to request services. A single LnkSvrMessage method call can contain one type of message request, and the type-dependent data for a message request is stored in the TRKSVR_MESSAGE_UNION structure. The type of the message is indicated in the **MessageType** field, and the message data is stored in the

MessageUnion field. The format of the **MessageUnion** field depends on the **MessageType** field. See section 2.2.11 for the definition of the TRKSVR_MESSAGE_TYPE enumeration used by the **MessageType** field. See section 3.1.4.1 for more information on the LnkSvrMessage method.

```
typedef struct {
  TRKSVR_MESSAGE_TYPE MessageType;
  TRKSVR_MESSAGE_PRIORITY Priority;
  [switch_is(MessageType)] union {
    [case(old_SEARCH)]
      old_TRKSVR_CALL_SEARCH old_Search;
    [case(MOVE_NOTIFICATION)]
      TRKSVR_CALL_MOVE_NOTIFICATION MoveNotification;
    [case(REFRESH)]
      TRKSVR_CALL_REFRESH Refresh;
    [case(SYNC_VOLUMES)]
      TRKSVR_CALL_SYNC_VOLUMES SyncVolumes;
    [case(DELETE_NOTIFY)]
      TRKSVR_CALL_DELETE Delete;
    [case(STATISTICS)]
      TRKSVR_STATISTICS Statistics;
    [case(SEARCH)]
      TRKSVR_CALL_SEARCH Search;
    [case(WKS_CONFIG)]
      TRKWKS_CONFIG WksConfig;
    [case(WKS_VOLUME_REFRESH)]
      unsigned long WksRefresh;
  };
  [string] WCHAR* ptszMachineID;
} TRKSVR_MESSAGE_UNION;
```

MessageType: The type of message to be selected from the TRKSVR_MESSAGE_TYPE enumeration. The value of this field indicates the format of the **MessageUnion** field.

Priority: The priority level of the operation. Valid values are defined in TRKSVR_MESSAGE_PRIORITY. Section 3.2.5.2 and 3.2.5.3, and subsections of section 3.2.6 specify how this field is set for different events.

(unnamed union): The message data for this message request. The **MessageType** field indicates which of these fields is used to format the data. (see section 2.2.11 for more information on the TRKSVR_MESSAGE_TYPE enumeration used by the **MessageType** field). The fields are defined as follows:

old_Search: Unused.

MoveNotification: If **MessageType** is MOVE_NOTIFICATION, this field contains message data for a MOVE_NOTIFICATION message, with the data formatted in the TRKSVR_CALL_MOVE_NOTIFICATION structure.

Refresh: If **MessageType** is REFRESH, this field contains message data for a REFRESH message, with the data formatted in the TRKSVR_CALL_REFRESH structure.

SyncVolumes: If **MessageType** is SYNC_VOLUMES, this field contains message data for a SYNC_VOLUMES message, with the data formatted in the TRKSVR_CALL_SYNC_VOLUMES structure.

Delete: If **MessageType** is DELETE_NOTIFY, this field contains message data for a DELETE_NOTIFY message, with the data formatted in the TRKSVR_CALL_DELETE structure.

Statistics: If **MessageType** is STATISTICS, this field contains message data for a STATISTICS message, with the data formatted in a TRKSVR_STATISTICS structure. This message type is not used in this protocol, but is included in this specification because it affects the size of the TRKSVR_MESSAGE_UNION structure as it is transmitted over the RPC Protocol.

Search: If **MessageType** is SEARCH, this field contains message data for a SEARCH message, with the data formatted in a TRKSVR_CALL_SEARCH structure.

WksConfig: Unused.

WksRefresh: Unused.

ptszMachineID: Unused. MUST be set to 0 and ignored on receipt.

2.2.12.1 TRKSVR_CALL_MOVE_NOTIFICATION

The TRKSVR_CALL_MOVE_NOTIFICATION structure is used in LnkSvrMessage method calls that specify a MOVE_NOTIFICATION message (see section 2.2.11), to indicate when one or more files have been moved off a volume. See section 3.2.6.1 for an example of the client using this structure.

```
typedef struct {
  unsigned long cNotifications;
  unsigned long cProcessed;
  SequenceNumber seq;
  long fForceSeqNumber;
  CVolumeId* pvolid;
  [size_is(cNotifications)] CObjId* rgobjidCurrent;
  [size_is(cNotifications)] CDomainRelativeObjId* rgdroidBirth;
  [size_is(cNotifications)] CDomainRelativeObjId* rgdroidNew;
} TRKSVR_CALL_MOVE_NOTIFICATION;
```

cNotifications: This field MUST contain the number of move notifications that were received.

cProcessed: On return to the client, this field MUST indicate the number of notifications from the request message that were actually processed.

seq: This field MUST be set by the client to the VolumeSequenceNumber for this VolumeID. This value is used by the client and server to detect whether or not notifications have been lost. For information about sequence numbering, see sections 3.1.4.2 and 3.2.4.2.

fForceSeqNumber: This field MUST be set by the client to indicate whether the **seq** value is to be ignored. If set, **seq** MUST be ignored. Sequence numbering is as specified in sections 3.1.4.2 and 3.2.4.2.

pvolid: This field MUST contain the VolumeID, which indicates the volume from which the files in this notification were moved. CVolumeId is as specified in [MS-DLTW] section 2.2.4.

rgobjidCurrent: This field MUST contain an array of ObjectIDs with the ObjectID for each file that was moved. CObjId is as specified in [MS-DLTW] section 2.2.5. Note that the previous FileLocation of each of the moved files MUST be determined by composing the VolumeID from the pvolid value with each of the entries in the rgobjidCurrent array.

rgdroidBirth: This field MUST contain an array of FileIDs with the FileID for each file that was moved in this request. Each element in the rgdroidBirth array corresponds to the entry with the same index in the rgobjidCurrent array. CDomainRelativeObjId is as specified in [MS-DLTW] section 2.2.3.

rgdroidNew: This field MUST contain an array of FileLocations, with the new FileLocation for each file that was moved in this request. Each element in the rgdroidNew array corresponds to the entry with the same index in the rgobjidCurrent and rgdroidBirth arrays.

2.2.12.2 TRKSVR_CALL_REFRESH

The TRKSVR_CALL_REFRESH structure is used in LnkSvrMessage method calls that specify a REFRESH message (see section 2.2.11), to indicate to the server that a file or volume is still in use. The server uses this to determine when an entry in its tables is no longer in use and can be deleted. See section 3.2.5.2 for an example of client use of this structure.

```
typedef struct {
  unsigned long cSources;
  [size_is(cSources)] CDomainRelativeObjId* adroidBirth;
  unsigned long cVolumes;
  [size_is(cVolumes)] CVolumeId* avolid;
} TRKSVR_CALL_REFRESH;
```

cSources: This field MUST contain the number of elements in the **adroidBirth** array.

adroidBirth: This field MUST contain an array of FileIDs, with the FileID for files on the client computer for which the client is requesting a refresh. The type of this field, CDomainRelativeObjId, is as specified in [MS-DLTW] section 2.2.3. If this array is empty, it indicates that there are no FileIDs to be refreshed by this request.

cVolumes: This field MUST contain the number of elements in the **avolid** array. Note that this is independent of the cSources value.

avolid: This field MUST contain an array of VolumeIDs for volumes on the client computer for which the client requests a refresh. CVolumeId is as specified in [MS-DLTW] section 2.2.4. If this array is empty, it indicates that there are no VolumeIDs to be refreshed by this request.

2.2.12.3 TRKSVR_CALL_SYNC_VOLUMES

The TRKSVR_CALL_SYNC_VOLUMES structure is used in LnkSvrMessage method calls that specify a SYNC_VOLUMES message (see section 2.2.11), to synchronize volumes between the client and server. For example, this structure is used by the client to request that the server create an entry in its ServerVolumeTable.

This structure holds an array of independent requests. Each of those individual requests is termed a subrequest in this protocol specification. See section 2.2.14 for more information about the individual subrequest data structures. See sections 3.2.5.3 and 3.2.6.4 for examples of the client using this structure.

```
typedef struct {
  unsigned long cVolumes;
  [size_is(cVolumes)] TRKSVR_SYNC_VOLUME* pVolumes;
} TRKSVR_CALL_SYNC_VOLUMES;
```

cVolumes: On input, the number of subrequests in this message. On return, this is the number of subrequests that the server processed. The details of this usage are specified in sections 3.1.4.4 and 3.2.4.4.

pVolumes: An array of subrequests.

2.2.12.4 TRKSVR_CALL_DELETE

The TRKSVR_CALL_DELETE structure is used in LnkSvrMessage method calls that specify a DELETE_NOTIFY message (see section 2.2.11), to indicate which files are to be removed from the FileTable. See section 3.2.6.2 for an example of the client using this structure.

```
typedef struct {
  unsigned long cdroidBirth;
  [size_is(cdroidBirth)] CDomainRelativeObjId* adroidBirth;
  unsigned long cVolumes;
  [size_is(cVolumes)] CVolumeId* pVolumes;
} TRKSVR_CALL_DELETE;
```

cdroidBirth: This field MUST contain the number of entries in the adroidBirth array.

adroidBirth: This field MUST contain an array of FileIDs of files that have been deleted. CDomainRelativeObjId is as specified in [MS-DLTW] section 2.2.3.

cVolumes: This field is unused and MUST be zero.

pVolumes: This field is unused and MUST be set to zero.

2.2.12.5 TRKSVR_STATISTICS

The TRKSVR_STATISTICS structure was originally defined for use in LnkSvrMessage method calls that specify a STATISTICS message. The STATISTICS message type is not used in this protocol, but is included in this specification because it affects the size of the TRKSVR_MESSAGE_UNION structure as it is transmitted over the RPC Protocol.

```
typedef struct {
  unsigned long cSyncVolumeRequests;
  unsigned long cSyncVolumeErrors;
  unsigned long cSyncVolumeThreads;
  unsigned long cCreateVolumeRequests;
  unsigned long cCreateVolumeErrors;
  unsigned long cClaimVolumeRequests;
  unsigned long cClaimVolumeErrors;
  unsigned long cQueryVolumeRequests;
  unsigned long cQueryVolumeErrors;
  unsigned long cFindVolumeRequests;
  unsigned long cFindVolumeErrors;
  unsigned long cTestVolumeRequests;
  unsigned long cTestVolumeErrors;
  unsigned long cSearchRequests;
  unsigned long cSearchErrors;
  unsigned long cSearchThreads;
  unsigned long cMoveNotifyRequests;
  unsigned long cMoveNotifyErrors;
  unsigned long cMoveNotifyThreads;
  unsigned long cRefreshRequests;
  unsigned long cRefreshErrors;
  unsigned long cRefreshThreads;
  unsigned long cDeleteNotifyRequests;
  unsigned long cDeleteNotifyErrors;
  unsigned long cDeleteNotifyThreads;
  unsigned long ulGCIterationPeriod;
  FILETIME ftLastSuccessfulRequest;
  HRESULT hrLastError;
  unsigned long dwMoveLimit;
  long lRefreshCounter;
  unsigned long dwCachedVolumeTableCount;
  unsigned long dwCachedMoveTableCount;
  FILETIME ftCachedLastUpdated;
  long fIsDesignatedDc;
  FILETIME ftNextGC;
  FILETIME ftServiceStart;
  unsigned long cMaxRPCThreads;
  unsigned long cAvailableRPCThreads;
  unsigned long cLowestAvailableRPCThreads;
  unsigned long cNumThreadPoolThreads;
```

```
  unsigned long cMostThreadPoolThreads;
  short cEntriesToGC;
  short cEntriesGCed;
  short cMaxDsWriteEvents;
  short cCurrentFailedWrites;
  struct {
    unsigned long dwMajor;
    unsigned long dwMinor;
    unsigned long dwBuildNumber;
  } Version;
} TRKSVR_STATISTICS;
```

2.2.12.6 TRKSVR_CALL_SEARCH

The TRKSVR_CALL_SEARCH structure is used in LnkSvrMessage method calls that specify a SEARCH message (see section 2.2.11), to query the DLT Central Manager server for the location of a file. See section 3.2.6.3 for an example of client use of this structure.

```
typedef struct {
  unsigned long cSearch;
  [size_is(cSearch)] TRK_FILE_TRACKING_INFORMATION* pSearches;
} TRKSVR_CALL_SEARCH;
```

cSearch: This value MUST be set to one.

pSearches: A pointer to a single search request. See TRK_FILE_TRACKING_INFORMATION (section 2.2.8).

2.2.12.7 old_TRKSVR_CALL_SEARCH

The old_TRKSVR_CALL_SEARCH structure is unused but is included in this protocol because it affects the definition of the TRKSVR_MESSAGE_UNION, as defined in section 2.2.13.

```
typedef struct {
  unsigned long cSearch;
  [size_is(cSearch)] old_TRK_FILE_TRACKING_INFORMATION* pSearches;
} old_TRKSVR_CALL_SEARCH;
```

2.2.12.8 TRKWKS_CONFIG

The TRKWKS_CONFIG structure is unused but is included in this protocol because it affects the definition of the TRKSVR_MESSAGE_UNION, as defined in section 2.2.13.

```
typedef struct {
  unsigned long dwParameter;
  unsigned long dwNewValue;
} TRKWKS_CONFIG;
```

2.2.13 TRKSVR_SYNC_TYPE

The TRKSVR_SYNC_TYPE enumeration specifies operations that can be handled during volume synchronization in a SYNC_VOLUMES message (see sections 2.2.11 and 2.2.12.3).

```
typedef [v1_enum] enum
{
  CREATE_VOLUME = 0,
```

```
  QUERY_VOLUME = 1,
  CLAIM_VOLUME = 2,
  FIND_VOLUME = 3,
  TEST_VOLUME = 4,
  DELETE_VOLUME = 5
} TRKSVR_SYNC_TYPE;
```

CREATE_VOLUME: Requests that the server maintain a mapping from a VolumeID to a VolumeSecret and MachineID. See section 3.2.5.3 for an example of this subrequest.

QUERY_VOLUME: Requests information about a VolumeID so that the client and server can keep their volume tables synchronized. See section 3.2.6.4 for an example of this subrequest.

CLAIM_VOLUME: Requests that a MachineID become recognized by the server as the VolumeOwner for a VolumeID. See section 3.2.5.3 for an example of this subrequest.

FIND_VOLUME: Requests the MachineID for a VolumeID. See section 3.2.6.4 for an example of this subrequest.

TEST_VOLUME: Reserved; MUST NOT be sent. Unused.

DELETE_VOLUME: Reserved; MUST NOT be sent. Unused.

2.2.14 TRKSVR_SYNC_VOLUME

The TRKSVR_SYNC_VOLUME structure is used as an array in calls to the LnkSvrMessage method that specifies a SYNC_VOLUMES message (see section 2.2.11), which in turn synchronizes volume information between the client and the server. Each TRKSVR_SYNC_VOLUME structure is termed a subrequest in this protocol specification.

```
typedef struct {
  HRESULT hr;
  TRKSVR_SYNC_TYPE SyncType;
  CVolumeId volume;
  CVolumeSecret secret;
  CVolumeSecret secretOld;
  SequenceNumber seq;
  FILETIME ftLastRefresh;
  CMachineId machine;
} TRKSVR_SYNC_VOLUME;
```

hr: A return value that indicates the success or failure of this TRKSVR_SYNC_VOLUME subrequest. The type of this field is an HRESULT, but unlike the standard definition, for this field, only zero is a successful return value. Except where otherwise specified, this value MUST NOT be TRK_E_VOLUME_QUOTA_EXCEEDED, which is defined in section 3.1.4.1. Any other nonzero value MUST be treated identically as a failure value.

SyncType: This indicates the type of synchronization request. Valid values are specified in section 2.2.13.

volume: The VolumeID to be synchronized. Whether this field is used depends on the SyncType value. For details, see sections 3.2.6.5 and 3.1.4.4. The CVolumeId type is as specified in [MS-DLTW] section 2.2.4.

secret: The new VolumeSecret to be used for this VolumeID. Whether this field is used depends on the SyncType value. For details, see sections 3.2.6.5 and 3.1.4.4.

secretOld: A VolumeSecret that is used to authenticate a VolumeOwner. Whether this field is used depends on the SyncType value. For details, see sections 3.2.6.5 and 3.1.4.4.

seq: A VolumeSequenceNumber that is used for synchronization of move notifications, as specified in section 3.1.4.2. Whether this field is used depends on the SyncType value. For details, see sections 3.2.6.5 and 3.1.4.4.

ftLastRefresh: The last time the server received a REFRESH notification from a client. Whether this field is used depends on the SyncType value. For details, see sections 3.2.6.5 and 3.1.4.4.

machine: A MachineID of a VolumeOwner. This VolumeOwner is the VolumeOwner for the VolumeID specified in the volume field. Whether this field is used depends on the SyncType value. For details, see sections 3.2.6.5 and 3.1.4.4; the CMachineId type is as specified in [MS-DLTW] section 2.2.2.

3 Protocol Details

This protocol is in the form of an RPC interface. The server of this interface responds to LnkSvrMessage requests.

3.1 DLT Central Manager Protocol Server Details

3.1.1 Abstract Data Model

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

The DLT Central Manager server receives requests with updates to information of linked files, or requests for information about linked files. The server maintains the following information:

CurrentRefreshTime: The current time, in units of days. This value MUST initially be zero the first time a server is initialized in a new domain.

RecentTableUpdateCount: A count of the number of updates that have recently been made to the **ServerVolumeTable** and **FileTable**, and a time stamp that indicates the last time this count was reset. When the server starts, it MUST set this time stamp to the current time. The **RecentTableUpdateCount** value MUST NOT exceed 1000. Every time in this protocol that the server checks whether or not this field is at its maximum value (1000) — and if the field is indeed at its maximum value—the server MUST do the following:

- Check if the current time is more than one hour past the last reset time.
- If the current time is more than one hour past the last reset time, then the server MUST update the **RecentTableUpdateCount** field to zero.
- Otherwise, the server MUST leave the **RecentTableUpdateCount** field at 1000.

ServerVolumeTable: A table with rows uniquely identified by a VolumeID, which contains the following fields:

- **VolumeID**: The VolumeID that is the key to this entry. The remaining fields provide information about this volume. When this VolumeID type is transmitted between the client and server as part of a request, the type is stored in a CVolumeId structure, as defined in section 2.2.5.
- **VolumeSequenceNumber**: Sequencing information that is used to coordinate notifications between the client and the server. When this type is transmitted between the client and server as part of a request, it is stored in a SequenceNumber structure, as defined in section 2.2.2.
- **VolumeSecret**: A value that is used to verify with the server that a particular volume is on a client computer. When this type is transmitted between the client and server as part of a request, it is stored in a CVolumeSecret structure, as defined in section 2.2.7.
- **VolumeOwner**: The MachineID that the server considers to be the owner of the VolumeID. When this type is transmitted between the client and server as part of a request, it is stored in a CMachineId structure, as defined in section 2.2.3.
- **RefreshTime**: A time stamp that indicates when this entry was last known to be correct. This field represents a current or previous value from the **CurrentRefreshTime** field described earlier in this section. When the RefreshTime type is transmitted between the client and server as part of a request, it is stored in a FILETIME structure as defined in [MS-DTYP] section 2.3.3.

FileTable: A table (with rows that are uniquely identified by a FileLocation or FileID) that contains the following fields:

- **PreviousFileLocation**: A FileLocation of the file prior to the FileLocation stored in the following FileLocation field. If the following FileID field is not present, then this field also represents the FileID. When this type is transmitted between the client and server as part of a request, it is stored in a CDomainRelativeObjId structure, as defined in section 2.2.4.
- **FileLocation**: The FileLocation of the file represented by this entry at some point in time after the value in the PreviousFileLocation. This is not necessarily the current FileLocation. When this type is transmitted between the client and server as part of a request, the type is stored in a CDomainRelativeObjId structure, as defined in section 2.2.4.
- **FileID**: The FileID of the file represented by this entry. This field might or might not be present in a given entry, as noted in the description of PreviousFileLocation. When this type is transmitted between the client and server as part of a request, the type is stored in a CDomainRelativeObjId structure, as defined in section 2.2.4.
- **RefreshTime**: A time stamp indicating when this entry was last known to be correct. This field represents a current or previous value from the **CurrentRefreshTime** field described earlier in this section. When this type is transmitted between the client and server as part of a request, the type is stored in a FILETIME structure as defined in [MS-DTYP] section 2.3.3.

The following is an example of the conceptual FileTable, where a file with FileID A moves to FileLocation B, and then to FileLocation C. (Only three fields are shown in the example.) The FileTable could have two entries at that point, such as the following.

PreviousFileLocation	FileLocation	FileID
A	B	(Unspecified)
B	C	A

Or the FileTable could have a single, equivalent entry, such as the following.

PreviousFileLocation	FileLocation	FileID
A	C	(Unspecified)

The DLT Central Manager protocol server also operates over a directory database of account information. That database is defined in [MS-SAMR], particularly the user object described in section 3.1.1, and the UserAccountControl described in sections 2.2.7.1 and 2.2.1.13.

Note The preceding conceptual data can be implemented by using a variety of techniques. Any data structure that stores the preceding conceptual data can be used in the implementation.

3.1.2 Timers

TableMaintenanceTimer: This timer is used for garbage collection of entries in the FileTable and ServerVolumeTable. This timer MUST execute on a period of 1 day, and when it fires it must be restarted. See section 3.1.5 for details on the processing that is performed when this timer expires.

3.1.3 Initialization

Parameters necessary to initialize the RPC Protocol are specified in section 2.1.

There MUST only be one instance of the DLT Central Manager server running on a computer, and it MUST run on all domain controllers of a domain.

3.1.4 Message Processing Events and Sequencing Rules

This section contains message processing events and sequencing rules.

Methods in RPC Opnum Order

Method	Description
LnkSvrMessage	Provides a way to send and receive messages to the DLT Central Manager server to query or update information. Opnum: 0
LnkSvrMessageCallback	Called by the DLT Central Manager server during a call to LnkSvrMessage. Opnum: 1

The methods MUST NOT throw an exception.

3.1.4.1 LnkSvrMessage (Opnum 0)

The LnkSvrMessage method provides a way to send and receive messages to the DLT Central Manager server to query or update information.

```
HRESULT LnkSvrMessage(
  [in] handle_t IDL_handle,
  [in, out] TRKSVR_MESSAGE_UNION* pMsg
);
```

IDL_handle: For information about the handle_t data type, see [MS-DTYP] section 2.1.3.

pMsg: Pointer to a message, in the format of a TRKSVR_MESSAGE_UNION structure. If this method fails, as indicated by a failure return value, the client MUST ignore any changes made by the server to this structure.

Return Values: See the following table and the explanation after it for more information on return values.

Exceptions Thrown: None.

The following table contains failure and success return values that have special behavior in this protocol. All failure values not listed in this table MUST be treated identically. Similarly, all success values not listed in this table MUST be treated identically. Except where otherwise stated, a return value MUST NOT be a value from this table. Except where otherwise specified, the server MUST return a success value.

Constant/value	Description
TRK_E_NOT_FOUND 0x8DEAD01B	A requested object was not found.
TRK_E_VOLUME_QUOTA_EXCEEDED 0x8DEAD01C	The server received a CREATE_VOLUME subrequest of a SYNC_VOLUMES request, but the **ServerVolumeTable** size limit for the RequestMachine value has already been reached.
TRK_E_SERVER_TOO_BUSY	The server is busy, and the client is to retry the request at a later

Constant/value	Description
0x8DEAD01E	time.
TRK_S_OUT_OF_SYNC 0x0DEAD100	The **VolumeSequenceNumber** of a MOVE_NOTIFICATION request is incorrect. See section 3.1.4.2.
TRK_S_VOLUME_NOT_FOUND 0x0DEAD102	The VolumeID in a request was not found in the server's **ServerVolumeTable**.
TRK_S_VOLUME_NOT_OWNED 0x0DEAD103	A notification was sent to the LnkSvrMessage method, but the RequestMachine for the request was not the VolumeOwner for a VolumeID in the request.
TRK_S_NOTIFICATION_QUOTA_EXCEEDED 0x0DEAD107	The server received a MOVE_NOTIFICATION request, but the **FileTable** size limit has already been reached.

The LnkSvrMessage method has only a single parameter, a union of type TRKSVR_MESSAGE_UNION (see section 2.2.12). But that union is defined to hold one of several types of requests, referred to in this protocol specification as messages. The message type for a given request is specified in the MessageType field of the TRKSVR_MESSAGE_UNION. The possible message types are defined in section 2.2.11. The formats of the different messages are defined in the sub-sections of section 2.2.12. The responses by the server to those different messages are specified in the remaining subsections of section 3.1.4, according to the **MessageType** field of the union.

Except where otherwise noted, the server that receives a request MUST ignore and leave unmodified all fields in the TRKSVR_MESSAGE_UNION structure of the *pMsg* parameter, as well as the structures referenced by the **MessageUnion** field of the TRKSVR_MESSAGE_UNION.

For security purposes, the server SHOULD restrict access to clients that are members of the authenticated users built-in security group. The client's identity is determined as described in [MS-RPCE], section 3.3.3.4.3.<4>

The TRKSVR_MESSAGE_UNION structure of the *pMsg* parameter contains a Priority field. The server MAY use this value to decide to reject some requests with a TRK_E_SERVER_TOO_BUSY return value, but it MUST NOT use this value to change the ordering of processing of messages from a caller.<5>

In this processing of this method call, the MachineID of the client that makes the request MUST be used as the RequestMachine value.<6>

Note During the processing of a LnkSvrMessage call, the server can call back to the client by using the LnkSvrMessageCallback method. See sections 3.1.4.4 and 3.2.4.1 for more information.

3.1.4.2 Receiving a MOVE_NOTIFICATION Message

The MOVE_NOTIFICATION message is received by the server as part of a LnkSvrMessage request, as defined in section 3.1.4.1. The server uses the information in this message to update the FileTable that is used to process SEARCH requests (see section 3.1.4.6) with information about the new locations of files.

This message consists of a TRKSVR_CALL_MOVE_NOTIFICATION structure (described in section 2.2.12.1), which contains information for zero or more move notifications.

If the RequestMachine, as specified in section 3.1.4.1, is not the VolumeOwner of the VolumeID indicated in the **pvolid** field of the request, then the request MUST return with a return value of TRK_S_VOLUME_NOT_OWNED. If the VolumeID does not exist in the ServerVolumeTable, the request MUST return with a value of TRK_S_VOLUME_NOT_FOUND.

If the **seq** field is not equal to the VolumeSequenceNumber for the VolumeID in the ServerVolumeTable, then:

- The message MUST return with a value of TRK_S_OUT_OF_SYNC.
- The **seq** field of the message MUST be updated by the server to be the value of the VolumeSequenceNumber field of the ServerVolumeTable for this volume.
- The server MUST NOT perform any of the following additional processing.

As stated earlier in this section, a single MOVE_NOTIFICATION message contains information for one or more files that have been moved. The fields of the TRKSVR_CALL_MOVE_NOTIFICATION structure for this message MUST be interpreted by the server as follows:

- The **pvolid** field MUST be interpreted as the VolumeID that identifies the volume from which all files in this message were moved.
- Each entry in the **rgobjidCurrent** array MUST be interpreted as the ObjectID of a file before it was moved. Therefore, as described in section 1.1, the file's PreviousFileLocation (the FileLocation before the move) is a composition of the preceding VolumeID and this ObjectID.
- Each corresponding entry in the rgdroidBirth array MUST be interpreted as the moved file's FileID.
- Each corresponding entry in the rgdroidNew array MUST be interpreted as the moved file's new FileLocation.

For example, if a MOVE_NOTIFICATION message contains two notifications, the information for the second notification is represented by the pvolid field, and by the second entry in each of the **rgobjidCurrent**, **rgdroidBirth**, and **rgdroidNew** arrays.

Each of these notifications MUST be processed by the server, in the order in which they appear in the array, as follows:

- If the **RecentTableUpdateCount** has reached its maximum value, this notification and all remaining notifications MUST NOT be processed, and the server MUST return a failure return value to the client.
- If the FileTable already contains an entry that maps the FileID to a FileLocation, and that FileLocation is the same value as the PreviousFileLocation specified in the notification, then the entry MUST be updated such that its FileLocation is the FileLocation specified in the notification. The **RecentTableUpdateCount** MUST also be incremented.
- Otherwise, if the FileTable has already reached its maximum size (defined in the following bulleted list), this notification and all remaining notifications MUST NOT be processed.
- Otherwise, a new entry MUST be added to the FileTable, with the PreviousFileLocation that was specified in the request stored in the PreviousFileLocation field, the new FileLocation from the request stored in the FileLocation field, and the FileID from the request stored in the FileID field. Also, the **RecentTableUpdateCount** MUST be incremented.

As stated earlier in this section, the FileTable has a maximum size. The maximum size of the FileTable MUST be calculated as follows:

- For every entry in the ServerVolumeTable up to 5000 entries, add 200 times the number of entries to the value for the FileTable size limit.
- For every entry in the ServerVolumeTable in addition to 5000 entries, add 100 times that number to the value for the FileTable size limit.

For example, if the ServerVolumeTable has 10 entries, the maximum FileTable size is 10*200, or 2000. If the ServerVolumeTable has 5010 entries, the maximum FileTable size is 5000*200 + 10*100, or 1,001,000.

If one or more notifications are not processed because the FileTable reaches its maximum size, the server MUST return a value of TRK_S_NOTIFICATION_QUOTA_EXCEEDED from the LnkSvrMessage request.

Note that this is a success code, so the server might still have processed some of the notifications in this request.

The server MUST increment the VolumeSequenceNumber for the VolumeID in the ServerVolumeTable once for each of the move notifications processed in this request. When the value of the VolumeSequenceNumber reaches its maximum positive value (2147483647), incrementing MUST result in its most negative value (2147483648). For example, if a MOVE_NOTIFICATION request has a **seq** field value of 10, a **cNotifications** field value of 3, and returns with a **cProcessed** field value of 2, the next MOVE_NOTIFICATION request for this VolumeID has a **seq** field value of 12.

3.1.4.3 Receiving a REFRESH Message

This message is received by the server as part of a LnkSvrMessage request, as defined in section 3.1.4.1.

The server uses the FileID and VolumeID information in this request to mark entries in its ServerVolumeTable and FileTable as active. Marking these entries as active prevents them from being deleted as part of the TableMaintenanceTimer handling, as specified in section 3.1.5.

The message consists of a TRKSVR_CALL_REFRESH structure (defined in section 2.2.12.2), which contains an array of zero or more FileIDs and an array of zero or more VolumeIDs.

The server MUST set the **cSources** and **cVolumes** fields to zero in the TRKSVR_CALL_REFRESH structure to zero.

For each FileID in the **adroidBirth** array field, if there is an entry in the FileTable whose FileID field is equivalent to that request FileID, and the **RecentTableUpdateCount** is less than its maximum value, then:

- The server MUST update the **RefreshTime** field for that entry in the FileTable with the value of the **CurrentRefreshTime**.
- The server MUST increment the **RecentTableUpdateCount**.

For each VolumeID in the **avolid** array field in this request, if there is an entry in the ServerVolumeTable for this volume, the **VolumeOwner** field of that entry equals the RequestMachine, and the **RecentTableUpdateCount** is less than its maximum value, then:

- The server MUST update the **RefreshTime** field with the value of the **CurrentRefreshTime**.
- The server MUST increment the **RecentTableUpdateCount**.

If the **RecentTableUpdateCount** is found to be at its maximum value above, the server MUST discontinue processing of the message, and MUST return to the client with a failure return value.

3.1.4.4 Receiving a SYNC_VOLUMES Message

This message is received by the server as part of a LnkSvrMessage request, as defined in section 3.1.4.1.

This request is used to synchronize volume information between the client and the server. This allows the client to create, query, modify, find, or delete one or more entries in the ServerVolumeTable.

The message consists of a TRKSVR_CALL_SYNC_VOLUMES structure (defined in section 2.2.12.3). This structure contains an array of TRKSVR_SYNC_VOLUME structures, each of which is termed a subrequest in this protocol specification. The **SyncType** field in the TRKSVR_SYNC_VOLUME structure identifies the specific subrequest. The remaining fields in this structure have different uses, depending on the **SyncType** field value.

Generally, for each subrequest, the server reads some of the fields in the TRKSVR_SYNC_VOLUME structure and sets other fields for return to the client, depending on the subrequest, as specified in the remainder of this section (and its subsections).

Unless otherwise specified, the server MUST set the **hr** field of the TRKSVR_SYNC_VOLUME structure in each subrequest to a failure value, and MUST ignore all other fields in this structure.

However, the server need not return the updated TRKSVR_SYNC_VOLUME structures to the client by returning from the LnkSvrMessage request. Instead, the server MAY<7> send the updated structures directly to the client, during the LnkSvrMessage call by calling the LnkSvrMessageCallback callback method (defined in section 3.2.4.1) and passing the updated TRKSVR_SYNC_VOLUME structures via the TRKSVR_MESSAGE_UNION field. In this case, the following MUST be followed:

- All subrequests MUST be processed by the server before calling the callback method.
- All of the TRKSVR_SYNC_VOLUME structures in the request MUST be sent to the client in the callback method. That is, the server MUST NOT return some of the updated structures in the callback and other of these structures in the return from the LnkSvrMessage request. Because of this, regardless of the return value from the LnkSvrMessageCallback method, the cVolumes field of the TRKSVR_CALL_SYNC_VOLUMES structure MUST be set to zero by the server before returning from the LnkSvrMessage request.
- The server MUST use the return value of the LnkSvrMessageCallback method as the return value for the LnkSvrMessage request.

In addition to the return value of the LnkSvrMessage method itself, there is an HRESULT type for each subrequest, in the **hr** field of the TRKSVR_SYNC_VOLUME structure.

The type of the **hr** field is HRESULT, but as defined in section 2.2.14, only zero is defined to be a successful return result. Unless otherwise specified in the following sections, the server MUST set the **hr** field of every subrequest to a value indicating failure.

The following sections define the handling by the server for each subrequest type. The subrequests MUST be processed in the order in which they appear in the TRKSVR_CALL_SYNC_VOLUMES structure. Except where otherwise noted in these sections, the server MUST ignore all fields of each TRKSVR_SYNC_VOLUME structure.

3.1.4.4.1 CLAIM_VOLUME

CLAIM_VOLUME is a subrequest of the SYNC_VOLUMES message request (see section 3.1.4.4). This type requests that the DLT Central Manager server update either the VolumeOwner or the VolumeSecret field of an entry in the ServerVolumeTable.

If **RecentTableUpdateCount** has reached its maximum value, the server MUST set the **hr** field of the TRKSVR_SYNC_VOLUME structure to TRK_E_SERVER_TOO_BUSY, and MUST NOT perform further processing of this subrequest.

For this subrequest, fields in the TRKSVR_SYNC_VOLUME structure have the following meaning:

- **volume**: Specifies the VolumeID of the volume for which an update is requested.

- **secret**: Specifies the VolumeSecret value of that volume.

If no entry exists in the ServerVolumeTable such that the VolumeID of the entry matches the volume field of the TRKSVR_SYNC_VOLUME structure, the server MUST set the **hr** field of that structure to a value indicating failure.

Otherwise, if such an entry does exist, but the value specified by the **secretOld** field of the TRKSVR_SYNC_VOLUME structure is not equal to the **VolumeSecret** field for that entry, and the RequestMachine is not equal to the VolumeOwner field for this entry, the server MUST set the **hr** field of the TRKSVR_SYNC_VOLUME structure to a value indicating failure.

Otherwise, the server MUST update the volume's entry in the ServerVolumeTable, and update the TRKSVR_SYNC_VOLUME structure, as follows:

- The server MUST increment the **RecentTableUpdateCount**.
- The server MUST set the **VolumeOwner** field of the entry with the RequestMachine value.
- The server MUST set the VolumeSecret field of the entry with the value from the **secret** field of the TRKSVR_SYNC_VOLUME structure.
- The server MUST set the **seq** field in the TRKSVR_SYNC_VOLUME structure with the value from the VolumeSequenceNumber field of the entry.
- The server MUST set the **ftLastRefresh** field in the TRKSVR_SYNC_VOLUME structure with the value of the **RefreshTime** field of the entry.
- The server MUST set the **hr** field of the TRKSVR_SYNC_VOLUME structure to zero.

3.1.4.4.2 FIND_VOLUME

FIND_VOLUME is a subrequest of the SYNC_VOLUMES message request (see section 3.1.4.4). This type requests that the DLT Central Manager server provide the VolumeOwner for a given VolumeID.

For the FIND_VOLUME subrequest, fields in the TRKSVR_SYNC_VOLUME structure have the following meaning:

- **volume**: Specifies the VolumeID for which the corresponding VolumeOwner is requested.

If the ServerVolumeTable contains no entry whose VolumeID is equal to the **volume** field from the TRKSVR_SYNC_VOLUME structure, then the server MUST set the hr field of the TRKSVR_SYNC_VOLUME structure to a value indicating failure.

Otherwise, the server MUST update the TRKSVR_SYNC_VOLUME structure as follows:

- The server MUST set the **machine** field to the value of the VolumeOwner field of the volume's entry in the ServerVolumeTable.
- The server MUST set the **hr** field to zero.

3.1.4.4.3 QUERY_VOLUME

QUERY_VOLUME is a subrequest of the SYNC_VOLUMES message request (see section 3.1.4.4). This type requests that the DLT Central Manager server provide the VolumeSequenceNumber for a given VolumeID.

For the QUERY_VOLUME subrequest, fields in the TRKSVR_SYNC_VOLUME structure have the following meaning:

- **volume**: Specifies the VolumeID for which the corresponding VolumeSequenceNumber is requested.

If the ServerVolumeTable contains no entry whose VolumeID is equal to the **volume** field from the TRKSVR_SYNC_VOLUME structure, then the server MUST set the **hr** field of the TRKSVR_SYNC_VOLUME structure to a value indicating failure.

Otherwise, the server MUST update the TRKSVR_SYNC_VOLUME structure as follows:

- The server MUST set the **seq** field with the value of the VolumeSequenceNumber from the volume's entry in the ServerVolumeTable.
- The server MUST set the **hr** field to zero.
- The server MAY set the **ftLastRefresh** field structure with the value of the RefreshTime from the volume's entry in the ServerVolumeTable.<8>

3.1.4.4.4 CREATE_VOLUME

CREATE_VOLUME is a subrequest of the SYNC_VOLUMES message request (see section 3.1.4.4). This type requests that the DLT Central Manager server create a new entry in its ServerVolumeTable.

If the **RecentTableUpdateCount** is at its maximum value, the server MUST set the **hr** field of the TRKSVR_SYNC_VOLUME structure to TRK_E_SERVER_TOO_BUSY, and MUST NOT process this subrequest any further.

For this subrequest, fields in the TRKSVR_SYNC_VOLUME structure have the following meaning:

- **secret**: The value that is to be used as the **VolumeSecret** field for the new entry in the ServerVolumeTable.

If there are already 26 entries in the ServerVolumeTable whose VolumeOwner is the RequestMachine, then the server MUST set the **hr** field of the TRKSVR_SYNC_VOLUME structure to TRK_E_VOLUME_QUOTA_EXCEEDED, and MUST NOT process this subrequest any further.

The server MUST generate a new VolumeID value. As described later in this section, this value is used as the VolumeID value for a new entry in the ServerVolumeTable. This value MUST be arbitrarily generated as any valid value, as defined in section 1.1. That is, the VolumeID MUST be a 16-byte binary value, the low-order bit of the first byte MUST be a zero, the value MUST NOT be all zeros, and the value MUST be unique within the ServerVolumeTable. This value also MUST be unique within this SYNC_VOLUMES message, for any other CREATE_VOLUME subrequest.

The fields of the TRKSVR_SYNC_VOLUME structure MUST be updated as follows:

- **volume**: MUST be set to the value of the VolumeID generated previously.
- **hr**: MUST be set to zero.

If the server does not use the LnkSvrMessageCallback method, the following steps MUST be followed. (The LnkSvrMessageCallback is defined in section 3.2.4.1, and the use by the server of that callback is defined in section 3.1.4.4.) If the server does use the LnkSvrMessageCallback method, the following steps MUST be followed by the server after calling the LnkSvrMessageCallback method, if that method returns from the client with a successful return value.

- The server MUST increment the **RecentTableUpdateCount**.
- The server MUST create a new entry in the ServerVolumeTable. The fields of this new entry MUST be set as follows:
  - **VolumeID**: This MUST be set to the VolumeID value created previously by the server.
  - **VolumeSequenceNumber**: This MUST be set to zero.

- **VolumeSecret**: This MUST be set to the value of the **secret** field of the TRKSVR_SYNC_VOLUME structure from the subrequest.
- **VolumeOwner**: This MUST be set to the RequestMachine of the request.
- **RefreshTime**: This MUST be set to the current value of the **CurrentRefreshTime**.

3.1.4.5 Receiving a DELETE_NOTIFY Message

The DELETE_NOTIFY message is received by the server as part of a LnkSvrMessage request, as defined in section 3.1.4.1.

The server uses the information in this message to update the FileTable used to process SEARCH requests (see section 3.1.4.6).

This message consists of a TRKSVR_CALL_DELETE structure (defined in section 2.2.12.4). In this structure, the **adroidBirth** array field represents an array of FileID values. In this message, the client notifies the server that each of the files associated with this FileID has been deleted.

For each of the FileID values, if each of the following is true:

- The **RecentTableUpdateCount** has not reached its maximum value.
- There is an entry in the FileTable whose PreviousFileLocation value is the FileID specified in this request.
- The VolumeID in the FileID matches the VolumeID of an entry in the ServerVolumeTable, and the VolumeOwner for that entry is equal to the RequestMachine. (As defined in section 1.1, a FileID has an embedded VolumeID value within it.)

then the entry found in the FileTable MUST be removed, and the **RecentTableUpdateCount** MUST be incremented. If as previously described the **RecentTableUpdateCount** reaches its maximum value, the server MUST NOT process any more FileID values, and MUST return to the client with a failure return value.

If the previous step does not result in a failure—that is, the **RecentTableUpdateCount** does not reach its maximum value—then the server MUST set the **cdroidBirth** field of the TRKSVR_CALL_DELETE structure to zero.

3.1.4.6 Receiving a SEARCH Message

The SEARCH message is received by the server as part of a LnkSvrMessage request, as defined in section 3.1.4.1.

The server responds to this message by calculating the current FileLocation and MachineID of the requested file.

This message consists of a TRKSVR_CALL_SEARCH structure (defined in section 2.2.12.6), which contains information about the file for which the client is searching. As defined in section 2.2.12.6, the TRKSVR_CALL_SEARCH structure consists of exactly one TRK_FILE_TRACKING_INFORMATION structure (defined in section 2.2.8).

The server MUST attempt to locate the primary entry in the FileTable for the FileLocation specified in this message, as follows:

- Look for an entry in the FileTable whose **PreviousFileLocation** field is that of the **droidLast** field of the TRK_FILE_TRACKING_INFORMATION structure in the message.
- If that entry is not found, look for an entry in the FileTable whose **PreviousFileLocation** field is that of the **droidBirth** field of the TRK_FILE_TRACKING_INFORMATION structure in the message.

If no entry in the FileTable can be found in this way, the server MUST set the **hr** field in the TRK_FILE_TRACKING_INFORMATION structure to a failure value, and MUST NOT perform further processing for this message.

Using the FileLocation for this entry, the server MUST check for an entry in the FileTable whose PreviousFileLocation is equal to that FileLocation. If such an entry is found, the server MUST repeat this step using that entry's FileLocation. This process MUST continue until no such subsequent entry is found.

For example, if the first entry found indicates that the file was moved from PreviousFileLocation A to FileLocation B, there might be another entry in the FileTable indicating that the file was moved from PreviousFileLocation B to FileLocation C. The preceding steps result in the server finding the entry with PreviousFileLocation of B, which represents the second move in this example.

Given this final FileLocation value (the FileLocation field of the last entry of C, found in the preceding example), the server MUST search for the entry in the ServerVolumeTable whose VolumeID field is equal to the VolumeID component of that FileLocation. (As defined in section 1.1, a FileLocation value has an embedded VolumeID value.) If found:

- The server MUST set the value of the **VolumeOwner** field of that entry into the **mcidLast** field of the TRK_FILE_TRACKING_INFORMATION request structure.
- The server MUST set the value of the final FileLocation into the **droidLast** field of the TRK_FILE_TRACKING_INFORMATION request structure.
- The server MUST set the **hr** field of the TRK_FILE_TRACKING_INFORMATION request structure to zero.

3.1.5 Timer Events

When the **TableMaintenanceTimer** expires, the server MAY<9> perform the following operations on its tables:

- For each entry in the ServerVolumeTable, if the difference between the **RefreshTime** field and the **CurrentRefreshTime** value is more than 90, then the entry MUST be deleted.
- Similarly, for each entry in the FileTable, if the difference between the **RefreshTime** field and the **CurrentRefreshTime** value is more than 90, then the entry MUST be deleted.
- The **CurrentRefreshTime** value MUST be incremented.

3.1.6 Other Local Events

There are no additional local events.

3.2 DLT Central Manager Protocol Client Details

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in the Distributed Link Tracking: Central Manager Protocol. This organization is described to explain how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that specified in this document.

The DLT Central Manager client monitors the volumes of the computer on which it runs, and sends information about volume and linked file updates to the DLT Central Manager server. The client also requests information from the server, such as a file's current FileLocation and current MachineID.

The DLT Central Manager client maintains the following information:

ClientVolumeTable: This table has an entry for each of the volumes of the computer on which it runs. <10>

VolumeInformation: This table maintains the following volume information for each volume in the ClientVolumeTable. This information MUST be stored on the volume itself. This allows the client to get information from a volume, if a volume is moved between clients (see section 3.2.6.6 for more information).

VolumeInformation contains the following fields:

- **VolumeID**: This value corresponds to an entry in the server's ServerVolumeTable. The client gets this value from the server by using a CREATE_VOLUME subrequest; see section 3.2.5.3 for more information.
- **VolumeSequenceNumber**: This is sequencing information that coordinates notifications between the client and the server. For an example of this field being used, see section 3.2.4.4.3.
- **VolumeSecret**: This value is used to verify with the server that a particular volume is on a client computer. For an example use of this field, see section 3.2.5.3 for details of the CLAIM_VOLUME subrequest.
- **VolumeOwner**: This is the MachineID that was last known to be the VolumeOwner of this volume, as defined by the VolumeOwner field of the ServerVolumeTable defined in section 3.1.1. For information on setting the VolumeOwner field, see sections 3.2.4.4.1 and 3.2.4.4.4.
- **VolumeTableQuotaExceeded**: This flag, either True or False, and initially set to False, indicates whether or not the client has recently received a return value from the server, which in turn indicates that this computer cannot create any new **VolumeID** values. For further details, see sections 3.2.4.4.4 and 3.2.5.4.
- **VolumeState**: This value indicates whether this volume is registered successfully in the server's ServerVolumeTable, defined in section 3.1.1. This variable can have one of three values: Owned, NotOwned, or NotCreated.
- **EnterNotOwnedTime**: This is a date and time value that indicates when the **VolumeState** value changed to NotOwned. The EnterNotOwnedTime value MUST be set if the **VolumeState** is NotOwned; otherwise, EnterNotOwnedTime MUST NOT be set.

MoveNotificationList: This is a list of records with information about files that have been moved from a volume on this machine. There MUST be one MoveNotificationList value for each volume in the ClientVolumeTable, and this table MUST in turn be stored on the volume itself. The entries in MoveNotificationList list MUST be in the order in which the files were moved (as described in section 3.2.6.1). Each entry MUST contain the ObjectID of the file, the FileID of the file, and the FileLocation of the file after it was moved. The client MAY keep as many entries in this list as it wants.<11>

Each entry in this list contains the following information (for information on the creation of an entry, see section 3.2.6.1):

- **MoveObjectID**: The ObjectID of the file before it was moved.
- **MoveFileID**: The FileID of the file.
- **MoveNewFileLocation**: The FileLocation of the file after the move.
- **MoveSequenceNumber**: The value of the VolumeSequenceNumber when this move notification was created.

MoveNotificationCursor: If this value exists, it indicates the next entry in the MoveNotificationList that is to be sent to the server in a MOVE_NOTIFICATION message. If this value does not exist, it indicates that all entries have been sent. See section 3.2.5.6 for more information.

MoveNotificationVolumeCursor: If this value exists, it indicates that a sequence of MOVE_NOTIFICATION messages is in progress, and indicates the current volume (from the ClientVolumeTable) for which notifications are being sent. See section 3.2.5.6 for more information.

CrossVolumeMoveFlag: This is a value of either zero or one, associated with a file, that indicates whether the file has been moved across volumes at any time in the past.

VolumeFileTable: This table contains FileInformation entries, and is maintained for each volume in the ClientVolumeTable. Like the ClientVolumeTable, this table MUST be stored on the volume itself. The VolumeFileTable has an entry for each file on the volume that is to be tracked with this protocol. Each entry has a **CrossVolumeMoveFlag** value, which MUST initially be set to zero.<12>

FileTableQuotaExceeded: This flag, either true or false, indicates whether or not the client has recently received a return value from the server that indicated that its FileTable had reached its maximum size. For further details, section 3.2.4.2 shows where this flag is set, section 3.2.5.4 shows where this flag is cleared, and section 3.2.6.1 shows where this flag is checked.

PendingDeleteNotificationList: This is a list of FileIDs for files that have been deleted from a **VolumeFileTable**, where the deletion occurred at least 5 minutes in the past. See section 3.2.5.5 for details on how items are added to this list, and section 3.2.4.5 for details on how items are removed from this list.

PrePendingDeleteNotificationList: This is a list of FileIDs for files that have been deleted from a **VolumeFileTable**, where the deletion occurred within the last 5 minutes. See section 3.2.6.2 for information on how items are added to this list, and section 3.2.4.5 on how items are removed from this list.

The DLT Central Manager protocol client also operates over a directory database of account information. That database is defined in [MS-SAMR], particularly the user object described in section 3.1.1, and the UserAccountControl bit field described in sections 2.2.7.1 and 2.2.1.13.

Note The preceding conceptual data can be implemented by using a variety of techniques. Any data structure that stores this conceptual data can be used in the implementation.

3.2.2 Timers

In general, the timers defined in this section run until they expire. After they expire, expiration processing is performed, as defined in section 3.2.5.

Some of these timers have features to randomly delay their expiration, to perform a retry behavior, and to persist across instantiations of the client. These behaviors are defined as follows:

- **Period**:

  Each timer described in this section has a period. A running timer MUST expire as soon as possible after both the period and the optional random delay (described as follows) have passed. A timer's period MUST NOT change.

- **Random delay**:

  If a timer defines a random delay, then after the period has passed, the timer MUST NOT expire until after the random delay. This is done so that if a server is servicing a large number of clients, the risk is reduced that all the clients will synchronize and flood the server with simultaneous requests. If a running timer has a delay, the delay MUST be randomly calculated after the timer's

period has passed. The means by which randomness is determined for the delay is arbitrary, but is bounded in the timer descriptions later in this section.

- **Persistence**:

  If a timer supports persistence, and the timer is running when the client is stopped, the timer's remaining period MUST be preserved across client stops and starts. For example, if the client starts a timer with a period of 60 minutes, and subsequently the client stops running for 40 minutes, then the timer period will expire 20 minutes after the client has been restarted. Note that only the period is persisted, not the random delay.

- **Retry Period**:

  If a timer supports retries, and a retry is required (as specified in section 3.2.5), it MUST be restarted to a retry delay, rather than stopping or restarting to its normal period. When the timer expires after this retry delay, the expiration steps of section 3.2.5 MUST again be followed. The length of the retry delay is defined in the following individual timer definitions.

Unless otherwise stated in the following timer descriptions, the only feature in the preceding list that timers MUST have is a period. Unless otherwise stated in the remainder of this section, timers MUST NOT restart automatically when they expire:

VolumeInitializationTimer: This timer is used by the client to synchronize its ClientVolumeTable with the server's ServerVolumeTable, and to synchronize its MoveNotificationList with the server's FileTable. This timer MUST have a period of 1 minute. If a retry of this timer is required (as specified in section 3.2.5.1), VolumeInitializationTimer MUST be restarted to a delay randomly selected between 30 minutes and 2 hours, but limited such that the timer does not exceed 6 hours past its original due time. Once this timer reaches 6 hours past its original due time, it MUST be stopped. See section 3.2.5.1 for details on the client actions that are taken when this timer expires.

RefreshTimer: This timer is used by the client to refresh the server state. This timer MUST have a period of 30 days, a random delay of up to 6 hours, persist, and be restarted automatically when it expires. If retries are required (as specified in section 3.2.4.3), this timer MUST be restarted to a delay randomly selected between 1 and 24 hours. See sections 3.2.5.2 for details on the client actions taken when this timer expires.

InfrequentMaintenanceTimer: This timer is used by the client to periodically validate/update VolumeInformation fields. This timer MUST have a period of 30 days, a random delay of up to 6 hours, persist, and be restarted automatically when it expires. See section 3.2.5.4 for details on the client actions taken when this timer expires.

FrequentMaintenanceTimer: This timer is used by the client to periodically validate and/or update VolumeInformation fields. This timer MUST have a period of 1 day, a random delay of up to 2 hours, persist, and be restarted automatically when it expires. See section 3.2.5.3 for details on the client actions taken when this timer expires.

DeleteNotificationTimer: This timer is used by the client to inform the server of files that have been deleted from the client. This timer MUST have a period of 5 minutes. See section 3.2.5.5 for details on the client actions taken when this timer expires.

MoveNotificationTimer: This timer is used by the client to inform the server of files that have been moved from a volume on the client to another volume (possibly to another volume on the client, or possibly to a volume on another computer). This timer MUST have a period of 30 seconds. If a retry is necessary, as specified in section 3.2.4.2, the retry delay MUST be calculated as follows:

- On the first retry, the delay MUST be 30 seconds.
- If subsequent retries are necessary, the retry delay MUST be the minimum of: double the previous retry delay; 4 hours; or a delay that would cause the sum of the retry delays to be

24 hours. Once this last condition is reached (that is, 24 hours and 30 seconds after the timer is started), the timer MUST be stopped.

3.2.3 Initialization

The client creates an RPC connection to the server by using the details specified in section 2.1 and with the following additional requirements:

- The client MUST establish a security context with an authentication level of Integrity (as defined in [MS-SPNG] section 3.1.1) for all messages except the SYNC_VOLUMES message when a SyncType type of CREATE_VOLUME or CLAIM_VOLUME is specified, as defined in section 3.2.5.3. In those cases, the client MUST use an authentication level of Confidentiality, also defined in section 3.1.1 of [MS-SPNG].
- The client MUST call a server that is running on a DC, within the client's domain, by using the domain controller locator specified in [MS-ADTS] section 6.3.
- The client MUST establish the RPC connection under a user account with a user principal name (UPN) in which the user account name is composed of the computer's MachineID, appended with a dollar sign ($). That user account must be a UF_SERVER_TRUST_ACCOUNT or a UF_WORKSTATION_TRUST_ACCOUNT. For more information on the user object and UserAccountControl bit field, see section 3.1.1.

When the client is started, it MUST create entries in the ClientVolumeTable for each volume that is to be tracked.<13> Initialization of those entries in the ClientVolumeTable is defined in section 3.2.6.6.

3.2.4 Message Processing Events and Sequencing Rules

As defined in section 3.1.4.1, requests sent with the LnkSvrMessage method are sent via the *pMsg* parameter, which is a TRKSVR_MESSAGE_UNION structure. In that structure is a **MessageType** field, defined by the TRKSVR_MESSAGE_TYPE enumeration, which indicates the format of the message data in the **MessageUnion** field.

Generally, requests are sent via fields on the TRKSVR_MESSAGE_UNION structure, and responses from the server are returned to the client by updating fields in this same structure. The client completion handling is covered in subsequent subsections for each type of message, according to the **MessageType** field of the TRKSVR_MESSAGE_UNION structure.

Except where otherwise stated, the client MUST ignore all fields in the TRKSVR_MESSAGE_UNION structure on completion of the LnkSvrMessage request.

3.2.4.1 LnkSvrMessageCallback (Opnum 1)

The LnkSvrMessageCallback method is an RPC callback method that provides a means for the DLT Central Manager server to call back to the client during a LnkSvrMessage call. As defined in section 3.1.4, this callback only occurs for SYNC_VOLUMES messages (for an example of this message, see section 3.2.5.3).

For more details on when this callback is used by the server, see section 3.1.4.4.

```
[callback] HRESULT LnkSvrMessageCallback(
  [in, out] TRKSVR_MESSAGE_UNION* pMsg
);
```

pMsg: Pointer to a message, in the format of a TRKSVR_MESSAGE_UNION structure. The **MessageType** field in this structure MUST be set to SYNC_VOLUMES.

Return Values: The return value is typed as an HRESULT, but for this method, a value of zero indicates success, and all other values indicate failure. Any nonzero value MUST be treated identically as a failure value.

The client MUST respond to this request by executing the steps in section 3.2.4.4 on each of the TRKSVR_SYNC_VOLUME structures in the TRKSVR_CALL_SYNC_VOLUMES structure within the *pMsg* parameter. In this way, the client is responding as though it received the updated structure in the completion of the LnkSvrMessage request.

If any subrequest indicates a failure—that is, if the **hr** field of any TRKSVR_SYNC_VOLUME structure is not zero—the client MUST return to the server with a return value that indicates failure.

For example, in a typical case where this callback method is used, processing proceeds in the following order:

- The client sends a SYNC_VOLUMES message to the server by calling LnkSvrMessage, as described, for example, in section 3.2.5.3.
- The server processes the request, updates the TRKSVR_CALL_SYNC_VOLUMES array in the request, and calls LnkSvrMessageCallback on the client.
- The client processes the subrequests in the updated TRKSVR_CALL_SYNC_VOLUMES array, as defined in section 3.2.4.4.
- The client returns from the LnkSvrMessageCallback method to the server.
- The server sets the **cProcessed** field of the TRKSVR_CALL_SYNC_VOLUMES structure to zero, and returns from the LnkSvrMessage method to the client.
- The client again performs the processing defined in section 3.2.4.4. But because the **cProcessed** field has been set to zero, the client takes no additional action, as defined in that section.

3.2.4.2 Completion of a MOVE_NOTIFICATION Message

See section 3.2.5.6 for information on the initiation of this request. Note in that definition that this MOVE_NOTIFICATION message is in reference to one volume, identified by the **pvolid** field of the TRKSVR_CALL_MOVE_NOTIFICATION structure of the request.

The processing performed by the client upon completion of the MOVE_NOTIFICATION message depends on the return value from the LnkSvrMessage call:

- **Zero**:

  As explained in 3.2.5.6, the **cNotifications** field in the TRKSVR_CALL_MOVE_NOTIFICATION structure of the request represents the number of notifications sent from the client to the server. Also, the **cProcessed** field in that structure, upon completion of the request, represents the number of notifications that were processed by the server (see section 3.1.4.2).

  The client MUST increment the MoveNotificationCursor value for the volume's MoveNotificationList by the number of entries indicated by **cProcessed**. If this increment moves the MoveNotificationCursor past the end of the entries in the MoveNotificationList, it MUST be cleared.

  If the **cProcessed** field is equal to the **cNotifications** field, the client MUST take the following steps:

  - If entries remain in this MoveNotificationList of the volume identified by the MoveNotificationVolumeCursor, then the client MUST initiate a new MOVE_NOTIFICATION message, following the steps in section 3.2.6.1.

- Otherwise, if the MoveNotificationVolumeCursor does not already reference the last entry in the ClientVolumeTable, then the client MUST advance the MoveNotificationVolumeCursor to the next entry in the ClientVolumeTable, and go back to the previous step.

- If neither of the previous two items is true, the client MUST clear the MoveNotificationVolumeCursor.

Note that, in the absence of failures, the preceding behavior can cause the client to send a series of MOVE_NOTIFICATION messages, such that a MOVE_NOTIFICATION message is sent for each of the entries of the MoveNotificationList of each of the volumes in the ClientVolumeTable.

If, however, the **cProcessed** field is less than the **cNotifications** field, then the client MUST follow the steps defined in the "Any other return value" entry at the end of this section.

- **TRK_S_OUT_OF_SYNC**:

  This return value indicates that the provided VolumeSequenceNumber did not match what the server was expecting, and that none of the notifications was processed. In this case, the **seq** field returned by the server in the TRKSVR_CALL_MOVE_NOTIFICATION structure of the request indicates the expected value.

  The client MUST respond as follows:

  - If the **MoveSequenceNumber** field of the entry in the MoveNotificationList referenced by the MoveNotificationCursor is less than the seq value returned by the server, then the client MUST send a new MOVE_NOTIFICATION message to the server, as specified in section 3.2.5.6, except that the client MUST set the **fForceSeqNumber** field in that MOVE_NOTIFICATION message to 1.

  - Otherwise, if there is an entry in the MoveNotificationList with a MoveSequenceNumber that matches the **seq** field, then the client MUST update the MoveNotificationCursor to reference that entry, and the client MUST send a new MOVE_NOTIFICATION message to the server, as specified in section 3.2.5.6.

  - If neither of the previous two items is true, the client MUST update the MoveNotificationCursor to reference the oldest entry in the MoveNotificationList, and MUST send a new MOVE_NOTIFICATION message to the server, as specified in section 3.2.5.6, except that the client MUST set the **fForceSeqNumber** field in that MOVE_NOTIFICATION message to 1.

- **TRK_S_VOLUME_NOT_OWNED or TRK_S_VOLUME_NOT_FOUND**:

  This return value indicates that the client computer is not the VolumeOwner of the VolumeID for the volume indicated in the request. If the VolumeState value for this volume is the Owned value, it MUST be set to NotOwned, and the EnterNotOwnedTime value MUST be set to the current date and time.

- **TRK_S_NOTIFICATION_QUOTA_EXCEEDED**:

  This return value indicates that the server's FileTable has reached its maximum size. The client MUST set its FileTableQuotaExceeded flag to True, as specified in section 3.2.1.

  The client also MUST increment the MoveNotificationCursor for the volume's MoveNotificationList by the number of entries indicated by the **cProcessed** field.

- **Any other return value**:

  If any other value is returned, and the MOVE_NOTIFICATION message was sent as part of the VolumeInitializationTimer processing described in section 3.2.5.1, then the client MUST put the VolumeInitializationTimer into its retry state, as described in section 3.2.2.

3.2.4.3 Completion of a REFRESH Message

If the LnkSvrMessage returns with a failure return value, the RefreshTimer MUST be restarted with a retry delay. See section 3.2.2 for information on the retry behavior of the RefreshTimer and its retry delay.

3.2.4.4 Completion of a SYNC_VOLUMES Message

The SYNC_VOLUMES message is composed of a TRKSVR_CALL_SYNC_VOLUMES structure, as described in section 2.2.12.3. That structure is composed of an array of TRKSVR_SYNC_VOLUME structures. Each TRKSVR_SYNC_VOLUME is termed a subrequest, and has a **SyncType** field that indicates the type of synchronization that the client requested. For example, a **SyncType** field of CREATE_VOLUME indicates that the client wants the server to create an entry in its ServerVolumeTable, and return a new VolumeID.

The **hr** field of the TRKSVR_SYNC_VOLUME structure indicates success or failure of the subrequest, as defined in section 2.2.14.

Unless otherwise specified, the client MUST ignore all fields of the TRKSVR_SYNC_VOLUME structure.

If the SYNC_VOLUMES message fails, or if the message succeeds but the **hr** field for a subrequest is TRK_E_SERVER_TOO_BUSY, and the message was sent as part of the VolumeInitializationTimer processing described in section 3.2.5.1, then the client MUST put the VolumeInitializationTimer into its retry state, as described in section 3.2.2.

If the SYNC_VOLUMES message succeeds but the **cProcessed** field of the TRKSVR_CALL_SYNC_VOLUMES structure is zero, the client MUST perform no further action.

If the SYNC_VOLUMES message succeeds, but the **hr** field for a subrequest does not succeed, then the client MUST ignore the fields of that subrequest, unless otherwise specified.

If the SYNC_VOLUMES message succeeds, the completion of each individual subrequest MUST be handled as specified in the following, corresponding sections.

3.2.4.4.1 CLAIM_VOLUME

If this subrequest fails, and the **VolumeState** field of the volume's VolumeInformation is set to Owned, the **VolumeState** MUST be updated to NotOwned.

Otherwise, the client MUST update the volume's VolumeInformation as follows:

- The VolumeSecret field MUST be set to be that of the **secret** field that was originally specified by the client in this request.
- The **RefreshTime** field MUST be set to that of the **ftLastRefresh** field of the TRKSVR_SYNC_VOLUME structure returned by the server.
- The VolumeOwner field MUST be set to the current machine's MachineID.
- The **VolumeState** field MUST be set to Owned.
- The **EnterNotOwnedTime** field MUST be cleared.

Additionally, if this subrequest succeeds, the client MUST check the **seq** field of the TRKSVR_SYNC_VOLUME subrequest returned by the server. If it differs from the VolumeSequenceNumber in the VolumeInformation maintained by the client for this volume, then the client MUST synchronize its VolumeSequenceNumber with the server (as specified in section 3.2.4.2) for the TRK_S_OUT_OF_SYNC processing.

3.2.4.4.2 FIND_VOLUME

The upper layer that triggered this call might use the **machine** field in its next step to locate the file. For example, the upper layer might initiate a LnkSearchMachine call on the Distributed Link Tracking: Workstation Protocol to the specified computer.

3.2.4.4.3 QUERY_VOLUME

If this subrequest fails, and the **VolumeState** field in the VolumeInformation for the volume is currently set to Owned, it MUST be updated to NotOwned, and the EnterNotOwnedTime MUST be set to the current time.

If the subrequest succeeds, and the **VolumeState** in the VolumeInformation of the volume is not currently set to Owned, then:

- The **VolumeState** field of the volume MUST be set to Owned.
- The **VolumeFileTable** field for the volume MUST be cleared.
- The **EnterNotOwnedTime** field MUST be cleared.

Additionally, if the subrequest succeeds and the **seq** field returned by the server does not match the VolumeSequenceNumber in the VolumeInformation for the volume, it is an indication that the server's FileTable is out of sync with the client's move notification records. The client MUST then synchronize with the server, as specified in section 3.2.4.2, for the TRK_S_OUT_OF_SYNC processing, which is also described in section 3.2.4.2.

3.2.4.4.4 CREATE_VOLUME

If this subrequest succeeds, the client MUST update the VolumeInformation for this volume as follows:

- Set the value in the **volume** field of the TRKSVR_SYNC_VOLUME structure into the VolumeID field of the VolumeInformation.
- Set the value in the **ftLastRefresh** field of the TRKSVR_SYNC_VOLUME structure into the **RefreshTime** field of the VolumeInformation.
- Set the **VolumeSequenceNumber** field to zero.
- Set the **VolumeState** field to Owned.
- Set the **VolumeOwner** field to the MachineID of the local machine.
- Clear the **VolumeFileTable** field.

If the subrequest fails with an error return value of TRK_E_VOLUME_QUOTA_EXCEEDED, the client MUST set the **VolumeTableQuotaExceeded** flag to TRUE.

3.2.4.5 Completion of DELETE_NOTIFY Message

If this request succeeds:

- Each of the FileIDs in the original message MUST be removed from the PendingDeleteNotificationList.
- If there are remaining entries in that list, the client MUST send another DELETE_NOTIFY message to the server, as specified in section 3.2.5.5.

- If there are no remaining entries, but there are entries in the PrePendingDeleteNotificationList, then all entries from the PrePendingDeleteNotificationList MUST be moved to the PendingDeleteNotificationList, and the DeleteNotificationTimer MUST be restarted.

If the request fails:

- The client MUST restart the DeleteNotificationTimer.
- All entries remaining in the PrePendingDeleteNotificationList MUST be removed. If there are entries in the PrePendingDeleteNotificationList, then all entries from the PrePendingDeleteNotificationList MUST be moved to the PendingDeleteNotificationList, and the DeleteNotificationTimer MUST be restarted.

3.2.4.6 Completion of a SEARCH Message

If this request succeeds, the client MUST return the resulting **FileLocation** and **MachineID** fields returned by the server to the upper layers that triggered the call.

3.2.5 Timer Events

3.2.5.1 VolumeInitializationTimer Expiration

When the VolumeInitializationTimer expires, the client MUST perform the same actions as are performed on expiration of the FrequentMaintenanceTimer, as defined in section 3.2.5.3. Whether or not the LnkSvrMessage processing in that step succeeds, the server MUST also perform the same actions as are performed on expiration of the MoveNotificationTimer, as defined in section 3.2.5.6.

3.2.5.2 RefreshTimer Expiration

When the RefreshTimer expires, the client MUST send zero or more REFRESH messages to make the server aware of FileIDs for files on the client's computer, and aware of VolumeIDs for volumes on the client's computer.

A REFRESH message is sent to the server by using a LnkSvrMessage request, as defined in section 3.1.4.1.

If the client's ClientVolumeTable and VolumeFileTable are both empty, the client MUST NOT send a REFRESH request to the server. Otherwise, one or more messages MUST be sent. The format and number of the messages sent are defined in the following paragraphs.

Each message is sent by specifying a MessageType in the TRKSVR_MESSAGE_UNION of REFRESH. This specification requires that the message include the TRKSVR_CALL_REFRESH structure, as described in section 2.2.12.2.

In each call, the client sets the **Priority** field of the TRKSVR_MESSAGE_UNION based on the timer's period and on when the timer was started. If the current time is greater than or equal to the timer's start time plus twice its period, then the **Priority** MUST be set to zero. Otherwise, the **Priority** MUST be set to a value calculated as follows, where all times are measured in seconds, and where division truncates any fractional part to return an integer result.

```
9 - ( (Current time - Timer start time + Timer period) * 10 ) / Timer period
```

The **adroidBirth** and **avolid** fields of the requests, and the number of requests, MUST be determined as follows:

- The first message MUST have an entry in the **avolid** array field for each entry in its ClientVolumeTable. The value of that entry in avolid MUST be that of the VolumeID from the VolumeInformation for that entry in the **ClientVolumeTable**. If the ClientVolumeTable is empty, the client MUST pass NULL for the value of the **avolid** field of the message (and zero for the corresponding **cVolumes** field).

- The first message also MUST have an entry in the adroidBirth array for up to 128 of the entries in its VolumeFileTable. The value of that entry in the **adroidBirth** field MUST be that of the FileID from the **FileInformation** for that entry in the VolumeFileTable. If all of the VolumeFileTables are empty, the client MUST pass NULL for the value of the **adroidBirth** field of the message (and zero for the corresponding **cSources** field).

- If there are more than 128 entries in the VolumeFileTables, and the previous message is successful (as defined in section 3.2.4.1), the client MUST repeat the previous step, followed by this step. Note that by this definition, subsequent to the first message, the **avoid** field of each message will be NULL (and the **cVolumes** field zero).

See section 3.1.4.3 for information on the server processing of the REFRESH message. See section 3.2.4.2 for information on the client's completion of that message.

3.2.5.3 FrequentMaintenanceTimer Expiration

When the FrequentMaintenanceTimer expires, the client takes the following actions.

If the **VolumeState** field of the VolumeInformation of any entry in the ClientVolumeTable is in the NotOwned state, and if the **EnterNotOwnedTime** field for that volume is more than 7 days prior to the current time, then the **VolumeState** field MUST be updated to a value of NotCreated, and the **EnterNotOwnedTime** field MUST be cleared.

If the **VolumeState** field of the VolumeInformation of any of the entries in the ClientVolumeTable is NotOwned or NotCreated and the **VolumeTableQuotaExceeded** field is not set, then the client MUST attempt to synchronize the volume(s) with the server by sending a SYNC_VOLUMES message to the server.

A SYNC_VOLUMES message is sent to the server by using a LnkSvrMessage request, as defined in section 3.1.4.1. In this request, the following fields MUST be set in the TRKSVR_MESSAGE_UNION parameter:

- The **MessageType** field MUST be set to SYNC_VOLUMES.

- The **Priority** field MUST be set to 6.

The TRKSVR_CALL_SYNC_VOLUMES structure of this request MUST have a TRKSVR_SYNC_VOLUME entry for each volume that is in the NotCreated or NotOwned state, where the **SyncType** field is set as follows:

- If the volume is in the NotCreated state, the **SyncType** field of the TRKSVR_SYNC_VOLUME structure MUST be set to CREATE_VOLUME. This is termed a CREATE_VOLUME subrequest.

- If the volume is in the NotOwned state, the **SyncType** field MUST be set to CLAIM_VOLUME. This is termed a CLAIM_VOLUME subrequest.

Unless otherwise specified, all fields of each TRKSVR_SYNC_VOLUME structure MUST be set to all zeros.

For a CLAIM_VOLUME subrequest, the client MUST specify the fields of the TRKSVR_SYNC_VOLUME structure as follows:

- **volume**: This field MUST be set to the value of the VolumeID from the volume's VolumeInformation.

- **secretOld**: This field MUST be set to the value of the VolumeSecret from the volume's VolumeInformation.
- **secret**: This field MUST be set to a new VolumeSecret generated randomly by the client.

(See section 3.1.4.4.1 for information on the server processing of a CLAIM_VOLUME subrequest. See section 3.2.4.4.1 for information on the client's completion of this subrequest.)

For a CREATE_VOLUME subrequest, the client MUST specify fields of the TRKSVR_SYNC_VOLUME structure as follows:

- **secret**: This field MUST be set to a new VolumeSecret generated randomly by the client.

(See section 3.1.4.4.4 for information on the server processing of a CREATE_VOLUME subrequest. See section 3.2.4.4.4 for information on the client's completion of this subrequest.)

3.2.5.4 InfrequentMaintenanceTimer Expiration

When the InfrequentMaintenanceTimer expires, the client MUST synchronize its ClientVolumeTable with the server as follows:

The client MUST send a SYNC_VOLUMES message request to the server. This is sent by calling LnkSvrMessage with a MessageType in the TRKSVR_MESSAGE_UNION (see section 2.2.12) of SYNC_VOLUMES, and a Priority value of zero.

In this message, the TRKSVR_CALL_SYNC_VOLUMES array (described in section 2.2.12.3) MUST have a TRKSVR_SYNC_VOLUME entry for each volume in the client's ClientVolumeTable. In each TRKSVR_SYNC_VOLUME structure:

- The **SyncType** field MUST be set to QUERY_VOLUME.
- The **volume** field MUST be set to the VolumeID from the VolumeInformation for that volume.
- The **seq** field MUST be set to the VolumeSequenceNumber from the VolumeInformation for that volume.
- All other fields MUST be set to all zeros.

See section 3.1.4.4.3 for the server processing of that subrequest. See section 3.2.4.4.3 for the client's completion of that subrequest.

After the completion of this message, as defined in section 3.2.4.4.3, the client MUST also subsequently do the following:

- Set the VolumeTableQuotaExceeded flag in the VolumeInformation for all entries of the ClientVolumeTable to False, and set the FileTableQuotaExceeded flag to False.
- Carry out the same steps as the steps performed for the FrequentMaintenanceTimer.

If any of the client's MoveNotificationCursor values are set, the client MUST attempt to send one or more MOVE_NOTIFICATION requests, as specified in section 3.2.6.1.

3.2.5.5 DeleteNotificationTimer Expiration

When the DeleteNotificationTimer expires, if there are entries in the PendingDeleteNotificationList, the client MUST send the server notifications of those deleted files. See section 3.2.6.2 for more information about the PendingDeleteNotificationList. This MUST be sent to the server in a DELETE_NOTIFY message, which is part of a LnkSvrMessage request, as defined in section 3.1.4.1. In this request, the MessageType of the TRKSVR_MESSAGE_UNION parameter (see section 2.2.12) MUST be set to DELETE_NOTIFY, and the **Priority** field MUST be set to 5.

If, at the time the client is to send the notification, it has multiple notifications to send, then the client MUST send a batch of as many notifications as possible, up to 32 notifications, in a single message (see section 2.2.12.4 for a description of the TRKSVR_CALL_DELETE structure). As described in section 3.2.4.4, if there are more than 32 entries in the PendingDeleteNotificationList, the client attempts to send repeated messages, so that a notification for each entry in the list is sent to the server.

In each DELETE_NOTIFY message, the client MUST specify the fields in the TRKSVR_CALL_DELETE structure used in this request as follows:

adroidBirth: A list of up to 32 FileIDs from the PendingDeleteNotificationList for files that have been deleted.

pVolumes: Reserved; this value MUST be set to a NULL pointer.

See section 3.1.4.5 for information on the server processing of this message. See section 3.2.4.4 for information on the client's completion of this message.

3.2.5.6 MoveNotificationTimer Expiration

When the timer expires, this expiration indicates to the client to check its tables to determine whether it is necessary to send file move notifications to the server by using a MOVE_NOTIFICATION message.

However, if the **FileTableQuotaExceeded** flag is set, the client MUST NOT perform any further processing defined in this section.

If the MoveNotificationVolumeCursor is not set, the client MUST set it to reference the first entry in the ClientVolumeTable.

Starting with the volume indicated by the MoveNotificationVolumeCursor, the client MUST find the first volume from the ClientVolumeTable that has a MoveNotificationList with a MoveNotificationCursor set, and whose VolumeState is set to Owned. If there is no such volume, then the client MUST clear the MoveNotificationVolumeCursor, and MUST NOT perform any further processing defined in this section. If there is such a volume, the MoveNotificationVolumeCursor MUST be updated to reference that entry in the ClientVolumeTable.

The client MUST send a MOVE_NOTIFICATION message to the server, using a LnkSvrMessage request, as defined in section 3.2.4.2. In this request, the MessageType of the TRKSVR_MESSAGE_UNION parameter (see section 2.2.12) MUST be set to MOVE_NOTIFICATION, and the Priority field MUST be set to 0. The client MUST also specify the following fields in the TRKSVR_CALL_MOVE_NOTIFICATION structure used in this request:

- **seq**: This field MUST be set to the value of the VolumeSequenceNumber from the VolumeInformation for the volume identified by the MoveNotificationVolumeCursor.
- **fForceSeqNumber**: This field MUST be set to zero, unless otherwise specified in section 3.2.4.2.
- **pvolid**: The field MUST be set to VolumeID from the VolumeInformation for the volume identified by the MoveNotificationVolumeCursor.
- **rgobjidCurrent**: This field MUST be set to a list of up to 32 ObjectIDs taken from the first entries of the MoveNotificationList.
- **rgdroidBirth**: This field MUST be set to a list of up to 32 FileIDs, taken from the entries of the MoveNotificationList that correspond to the values in the **rgobjidCurrent** field.
- **rgdroidNew**: This field MUST be set to a list of up to 32 FileLocations taken from the entries of the MoveNotificationList that correspond to the values in the **rgobjidCurrent** and **rgdroidBirth** fields.

- **cProcessed**: This field MUST be set to zero.

See section 3.1.4.2 for information about the server processing of this message. See section 3.2.4.2 for information about the client's completion of this request.

3.2.6 Other Local Events

3.2.6.1 A File Has Been Moved

When a file in the client's VolumeFileTable is moved to the VolumeFileTable of the same client or of a different client, the following steps MUST be performed:

- An entry MUST be created in the MoveNotificationList for the volume on which the file was located before the move, as defined in section 3.2.1, and the following information about the moved file MUST be put into that entry: the file's ObjectID before the move, the file's FileID, and the file's FileLocation after the move.
- The MoveNotificationTimer for that client MUST be started (if it is not already running).
- An entry MUST be created in the VolumeFileTable for the volume to which the file was moved.
- The **CrossVolumeMoveFlag** MUST be set to 1 in that entry.

3.2.6.2 A File Has Been Deleted

If a file in a VolumeFileTable is deleted, the file's **CrossVolumeMoveFlag** value is set to 1, and the PrePendingDeleteNotificationList has no more than 5000 entries in it, then the file's FileID MUST be added to the PrePendingDeleteNotificationList. If the DeleteNotificationTimer is not already running, it MUST be started.

3.2.6.3 A File Cannot Be Found

When a file has been moved, and the client wants to find it again, it MUST send a SEARCH message to the server. This message is sent to the server by using a LnkSvrMessage request, as defined in section 3.1.4.1, which specifies a MessageType in the TRKSVR_MESSAGE_UNION of SEARCH and a Priority value of zero.

The client MUST also specify the following fields in the TRK_FILE_TRACKING_INFORMATION structure used in this request:

- The **droidBirth** field MUST be set to the FileID of the file that is being sought.
- The **droidLast** field MUST be set to the last known FileLocation for the file.
- The **mcidLast** field MUST be set to zero.
- The **hr** field MUST be set to zero.

For example, these **droidBirth** and **droidLast** values could come from the FileLinkInformation maintained by a client of the Distributed Link Tracking: Workstation Protocol, as specified in [MS-DLTW] section 3.2.1.

See section 3.1.4.6 for information about the server processing of this message. See section 3.2.4.5 for the client's completion of this message.

3.2.6.4 A Volume Cannot Be Found

When a client needs to determine the MachineID for a given VolumeID, the client MUST use the FIND_VOLUME message. This message MUST be sent to the server by using a LnkSvrMessage request, as defined in 3.1.4.1. In that request, the client MUST specify a MessageType in the TRKSVR_MESSAGE_UNION of SYNC_VOLUMES, set the Priority value to zero, and set the **SyncType** field within the TRKSVR_CALL_SYNC_VOLUMES structure to FIND_VOLUME.

For example, this request might be sent if a client is searching for a file and received a TRK_E_REFERRAL error from the LnkSearchMachine call to a DLT Workstation server (as specified in [MS-DLTW]), and where a call to the DLT Central Manager returned TRK_E_NOT_FOUND. The client might then use the VolumeID component of the *fileLocationNext* parameter of that request in a FIND_VOLUME request as the **volume** field.

The client MUST also specify the following field in the TRKSVR_SYNC_VOLUME structure:

- The **volume** field MUST be set to the VolumeID whose MachineID is desired.
- All other fields of the TRKSVR_SYNC_VOLUME structure MUST be set to all zeros.

See section 3.1.4.4.2 for information about the server processing of this subrequest. See section 3.2.4.4.2 for information about the client's completion of this subrequest.

3.2.6.5 The Client Changes Domains

As defined in section 3.2.3, a client uses a server that is running on a DC in the client's domain. If the client enters a new domain, it MUST restart the VolumeInitializationTimer if that timer is not already running or if that timer is performing retry operations, as described in section 3.2.5.1. Also, for any of the client's volumes for which the **VolumeState** field of the VolumeInformation is set to Owned, the **VolumeState** MUST be changed to NotOwned.

3.2.6.6 A New Volume is Discovered

During initialization, as defined in section 3.2.3, the client discovers volumes to be added to the ClientVolumeTable. The client MAY<14> also discover such volumes after initialization.

When a volume is added to the ClientVolumeTable, if there is no VolumeInformation associated with the volume, the VolumeInformation MUST<15> be initialized. The VolumeInformation for that volume MUST be initialized as follows:

- The **VolumeID** field MUST be set to a GUID that is a valid VolumeID value. That is, the **VolumeID** field MUST be set to a value with a low statistical likelihood of being duplicated, as defined in section 1.1, such that it is unique across the volumes on the current machine, does not consist of all zeros, and has a low order bit of the first byte at zero.
- The **VolumeSecret** field MUST be initialized to all zeros.
- The **VolumeSequenceNumber** field MUST be initialized to zero.
- The **VolumeOwner** field MUST be set to the MachineID of the current machine.
- The **VolumeTableQuotaExceeded** field flag MUST be initialized to False.
- The **VolumeState** field MUST be initialized to NotCreated, and the **EnterNotOwnedTime** field MUST be left unset.

Additionally, in this case, the client MUST clear the VolumeFileTable.

If, during initialization, the VolumeInformation associated with the volume does exist but the **VolumeOwner** field is not the current machine's MachineID, then the VolumeInformation requires an

update. This situation can happen, for example, if a volume is physically moved between machines. The client MUST do the following:

- The **VolumeOwner** field MUST be set to the current machine's MachineID.
- If the **VolumeState** field is set to Owned, it MUST be updated to NotOwned, and the **EnterNotOwnedTime** field MUST be set to the current time.

4 Protocol Examples

This example shows a sample call, following the example specified in section 1.3, with IDs set as follows:

- O1: 6479f083cfb245c29c713f586d6e038f
- V1: 9d7e9c15f59b4cf9952b03616aa51ebe
- O3: 20e435b512f64c848a1acd8737359b24
- V3: 61ac933f7d2546149715c9d928b23f5e
- M3: ef073687d9984f8c974b0d7e9a4bba48

The client sends a LnkSvrMessage message with a TRKSVR_CALL_SEARCH structure as follows.

```
typedef struct {
    TRKSVR_MESSAGE_TYPE MessageType = SEARCH;
    TRKSVR_MESSAGE_PRIORITY Priority = PRI_9;

    [switch_is(MessageType)] union {
        ...
        [case (SEARCH)]
        TRKSVR_CALL_SEARCH Search = {see below};
    };

    [string] TCHAR *ptszMachineID = NULL;
} TRKSVR_MESSAGE_UNION;

typedef struct {
    [range(1,1)] unsigned long cSearch = 1;
    [size_is(cSearch)] TRK_FILE_TRACKING_INFORMATION *rgSearch =
        {see below};
} TRKSVR_CALL_SEARCH;

typedef struct {
    CDomainRelativeObjId droidBirth= {
            9d7e9c15f59b4cf9952b03616aa51ebe,
            6479f083cfb245c29c713f586d6e038f };
    CDomainRelativeObjId droidLast = {
            irrelevant, filled in by server};
    CMachineId mcidLast = {
            irrelevant, filled in by server};
    HRESULT hr = {
            irrelevant, filled in by server};
} TRK_FILE_TRACKING_INFORMATION;
```

The server looks up the specified FileID, finds an entry in its database, returns zero from the LnkSvrMessage method, with the TRKSVR_MESSAGE_UNION and TRKSVR_CALL_SEARCH wrapper left unmodified and the embedded TRK_FILE_TRACKING_INFORMATION updated as follows.

```
typedef struct {
    CDomainRelativeObjId droidBirth = {
            9d7e9c15f59b4cf9952b03616aa51ebe,
            6479f083cfb245c29c713f586d6e038f };
    CDomainRelativeObjId droidLast = {
            61ac933f7d2546149715c9d928b23f5e,
            20e435b512f64c848a1acd8737359b24 };
    CMachineId mcidLast = {
            ef073687d9984f8c974b0d7e9a4bba48 };
    HRESULT hr = 0;
```

```
} TRK_FILE_TRACKING_INFORMATION;
```

5 Security

5.1 Security Considerations for Implementers

There are no special security considerations for implementers.

5.2 Index of Security Parameters

Security parameter	Section
Authenticated users group	3.1.4.1
Authentication service parameters	3.1.3 and 3.2.3

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided as follows, where "ms-dtyp.idl" is the IDL found in [MS-DTYP], Appendix A.

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

typedef signed long SequenceNumber;



typedef struct CVolumeSecret {
    byte _abSecret[8];
} CVolumeSecret;



typedef struct {
    WCHAR tszFilePath[ 257 ];
    CDomainRelativeObjId droidBirth;
    CDomainRelativeObjId droidLast;
    HRESULT hr;
} old_TRK_FILE_TRACKING_INFORMATION; // Unused



typedef struct {
    CDomainRelativeObjId droidBirth;
    CDomainRelativeObjId droidLast;
    CMachineId mcidLast;
    HRESULT hr;
} TRK_FILE_TRACKING_INFORMATION;



typedef struct {
    unsigned long cSearch;
    [size_is(cSearch)]
        old_TRK_FILE_TRACKING_INFORMATION *pSearches;
} old_TRKSVR_CALL_SEARCH; // Unused



typedef struct {
    unsigned long cSearch;
    [size_is(cSearch)]
        TRK_FILE_TRACKING_INFORMATION *pSearches;
} TRKSVR_CALL_SEARCH;



typedef struct {
    unsigned long cNotifications;
    unsigned long cProcessed;
    SequenceNumber seq;
    long fForceSeqNumber;
    CVolumeId *pvolid;
    [size_is(cNotifications)]
        CObjId *rgobjidCurrent;
    [size_is(cNotifications)]
        CDomainRelativeObjId *rgdroidBirth;
    [size_is(cNotifications)]
        CDomainRelativeObjId *rgdroidNew;
} TRKSVR_CALL_MOVE_NOTIFICATION;
```

```
typedef struct {
    unsigned long cSources;
    [size_is(cSources)]
        CDomainRelativeObjId *adroidBirth;
    unsigned long cVolumes;
    [size_is(cVolumes)]
        CVolumeId *avolid;
} TRKSVR_CALL_REFRESH;



typedef struct {
    unsigned long cdroidBirth;
    [size_is(cdroidBirth)]
        CDomainRelativeObjId * adroidBirth;
    unsigned long cVolumes;
    [size_is(cVolumes)]
        CVolumeId *pVolumes;
} TRKSVR_CALL_DELETE;



typedef [v1_enum] enum {
    CREATE_VOLUME = 0,
    QUERY_VOLUME = 1,
    CLAIM_VOLUME = 2,
    FIND_VOLUME = 3,
    TEST_VOLUME = 4, // Unused
    DELETE_VOLUME = 5 // Unused
} TRKSVR_SYNC_TYPE;



typedef struct {
    HRESULT hr;

    TRKSVR_SYNC_TYPE SyncType;
    CVolumeId volume;
    CVolumeSecret secret;
    CVolumeSecret secretOld;
    SequenceNumber seq;
    FILETIME ftLastRefresh;
    CMachineId machine;
} TRKSVR_SYNC_VOLUME;


typedef struct {
    unsigned long cVolumes;
    [size_is(cVolumes)]
        TRKSVR_SYNC_VOLUME * pVolumes;
} TRKSVR_CALL_SYNC_VOLUMES;



typedef struct {
  unsigned long cSyncVolumeRequests;
  unsigned long cSyncVolumeErrors;
  unsigned long cSyncVolumeThreads;
  unsigned long cCreateVolumeRequests;
  unsigned long cCreateVolumeErrors;
  unsigned long cClaimVolumeRequests;
  unsigned long cClaimVolumeErrors;
  unsigned long cQueryVolumeRequests;
  unsigned long cQueryVolumeErrors;
  unsigned long cFindVolumeRequests;
  unsigned long cFindVolumeErrors;
  unsigned long cTestVolumeRequests;
  unsigned long cTestVolumeErrors;
  unsigned long cSearchRequests;
```

```
  unsigned long cSearchErrors;
  unsigned long cSearchThreads;
  unsigned long cMoveNotifyRequests;
  unsigned long cMoveNotifyErrors;
  unsigned long cMoveNotifyThreads;
  unsigned long cRefreshRequests;
  unsigned long cRefreshErrors;
  unsigned long cRefreshThreads;
  unsigned long cDeleteNotifyRequests;
  unsigned long cDeleteNotifyErrors;
  unsigned long cDeleteNotifyThreads;
  unsigned long ulGCIterationPeriod;
  FILETIME ftLastSuccessfulRequest;
  HRESULT hrLastError;
  unsigned long dwMoveLimit;
  long lRefreshCounter;
  unsigned long dwCachedVolumeTableCount;
  unsigned long dwCachedMoveTableCount;
  FILETIME ftCachedLastUpdated;
  long fIsDesignatedDc;
  FILETIME ftNextGC;
  FILETIME ftServiceStart;
  unsigned long cMaxRPCThreads;
  unsigned long cAvailableRPCThreads;
  unsigned long cLowestAvailableRPCThreads;
  unsigned long cNumThreadPoolThreads;
  unsigned long cMostThreadPoolThreads;
  short cEntriesToGC;
  short cEntriesGCed;
  short cMaxDsWriteEvents;
  short cCurrentFailedWrites;
  struct {
    unsigned long dwMajor;
    unsigned long dwMinor;
    unsigned long dwBuildNumber;
  } Version;
} TRKSVR_STATISTICS;



typedef struct {
    unsigned long dwParameter;
    unsigned long dwNewValue;
} TRKWKS_CONFIG;



typedef [v1_enum] enum {
    old_SEARCH,
    MOVE_NOTIFICATION = 1,
    REFRESH = 2,
    SYNC_VOLUMES = 3,
    DELETE_NOTIFY = 4,
    STATISTICS = 5,
    SEARCH = 6,
    WKS_CONFIG, // Unused
    WKS_VOLUME_REFRESH // Unused
} TRKSVR_MESSAGE_TYPE;



typedef [v1_enum] enum {
    PRI_0=0,
    PRI_1=1,
    PRI_2=2,
    PRI_3=3,
    PRI_4=4,
    PRI_5=5,
    PRI_6=6,
    PRI_7=7,
```

```
        PRI_8=8,
        PRI_9=9
    } TRKSVR_MESSAGE_PRIORITY;



    typedef struct {
        TRKSVR_MESSAGE_TYPE MessageType;
        TRKSVR_MESSAGE_PRIORITY Priority;
        [switch_is(MessageType)] union {
            [case (old_SEARCH)]
                old_TRKSVR_CALL_SEARCH old_Search; // Unused
            [case (MOVE_NOTIFICATION)]
                TRKSVR_CALL_MOVE_NOTIFICATION MoveNotification;
            [case (REFRESH)]
                TRKSVR_CALL_REFRESH Refresh;
            [case (SYNC_VOLUMES)]
                TRKSVR_CALL_SYNC_VOLUMES SyncVolumes;
            [case (DELETE_NOTIFY)]
                TRKSVR_CALL_DELETE Delete;
            [case (STATISTICS)]
                TRKSVR_STATISTICS Statistics;
            [case (SEARCH)]
                TRKSVR_CALL_SEARCH Search;
            [case (WKS_CONFIG)]
                TRKWKS_CONFIG WksConfig; // Unused
            [case (WKS_VOLUME_REFRESH)]
                unsigned long WksRefresh; // Unused
        };
        [string] WCHAR *ptszMachineID; // Unused
    } TRKSVR_MESSAGE_UNION;



    [
        uuid(4da1c422-943d-11d1-acae-00c04fc2aa3f),
        version(1.0),
        pointer_default(unique)
    ]

    interface trksvr {
        HRESULT
        LnkSvrMessage (
            [in] handle_t IDL_handle,
            [in, out] TRKSVR_MESSAGE_UNION * pMsg
        );

        [callback]
        HRESULT
        LnkSvrMessageCallback(
            [in, out] TRKSVR_MESSAGE_UNION * pMsg
        );
    }
```

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 2000 operating system
- Windows Server 2003 operating system
- Windows XP operating system

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

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

<1> Section 1.3: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: The user on M0 indicates a desire to keep information about F1.txt by creating a Shell shortcut. The Shell shortcut is a file stored on M0 that holds the UNC, MachineID, FileLocation, and FileID for F1.txt.

<2> Section 1.3: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: The user initiates a request to open the file by activating the Shell shortcut that was created to store information about the file, F1.txt.

<3> Section 1.3: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: After storing the new FileLocation and UNC in the Shell shortcut file, the Shell shortcut implementation opens the file; for example, the shortcut implementation loads the file into the Notepad program. None of this processing is visible to end users. End users simply activate the Shell shortcut by double-clicking it, and then view the contents of the file in Notepad.

<4> Section 3.1.4.1: Windows Server 2003 implements this restriction.

<5> Section 3.1.4.1: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: When the server is busy processing several requests, it uses the priority field to determine which requests to process and which requests to reject with a TRK_E_SERVER_TOO_BUSY return value.

<6> Section 3.1.4.1: For example, in all versions of Windows listed in the supported products list in Appendix B: Product Behavior, the RequestMachine is determined as follows:

- The client identity is obtained, as described in [MS-RPCE], section 3.3.3.4.3, and the client identity is used to obtain the user principal name (UPN).
- That client identity is used to access the client's user object, described in section 3.1.1.
- If that user object has a UserAccountControl with neither UF_SERVER_TRUST_ACOUNT nor UF_WORKSTATION_TRUST_ACCOUNT set, or if the user account name of the UPN does not end in a dollar sign ($), the request fails. For more information on the user object and UserAccountControl, see section 3.1.1.
- The post-fix dollar sign ($) is removed from the user account name of the UPN, and then the name is used as the value of the RequestMachine.

<7> Section 3.1.4.4: Windows 2000: The protocol uses the LnkSvrMessageCallback to return information to the client if the SYNC_VOLUMES request includes a CREATE_VOLUME subrequest.

Windows Server 2003: The callback is not used; information is provided to the client on return from the LnkSvrMessage request.

<8> Section 3.1.4.4.3: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: The server sets the **ftLastRefresh** field of the structure with the value of the **RefreshTime** from the volume's entry in the ServerVolumeTable. But as specified in section 3.2.4.4.3, the client does not process this value.

<9> Section 3.1.5: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: This behavior is performed. If there are multiple instances of the server running, because there are multiple DCs in the domain, this behavior is only performed by one instance.

<10> Section 3.2.1: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: All local, compatible volumes are initialized, up to 26 volumes. If there are more than 26 compatible volumes, 26 are chosen arbitrarily (the first 26 compatible volumes returned by FindFirstVolume as described in [MSDN-FNDFSTVOL] and FindNextVolume as described in[MSDN-FNDNXTVOL]).

A volume is considered compatible if all of the following are true:

- The volume is formatted with the NTFS file system.
- The file system supports ObjectIDs.

  This is determined by using the GetVolumeInformation request, and checking in the *lpFileSystemFlags* parameter to GetVolumeInformation that the FILE_SUPPORTS_OBJECT_IDS flag is set. For more information on the GetVolumeInformation request, see [MSDN-GETVOLINFO].
- The volume is nonremovable.

  This is determined by using the GetDriveType request, and checking that the return value is DRIVE_FIXED. For more information on the GetDriveType request, see [MSDN-GETDRIVETYPE].

<11> Section 3.2.1: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: The client maintains 10,000 entries in this list. When this list is its maximum size, and a new entry is to be added, the entry is added such that it replaces the oldest entry in the list.

<12> Section 3.2.1: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: A file is considered to be part of the VolumeFileTable if it has an ObjectID. For example, a file has an ObjectID if a FSCTL_CREATE_OR_GET_OBJECT_ID request ([MS-FSCC], section 2.3.1) is successfully processed for the file. This request is sent for a file if a Shell Link is created for it. For more information on Shell Links, see [MSDN-SHELLLINKS].

<13> Section 3.2.3: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: All local, compatible volumes are initialized, up to 26 volumes. If there are more than 26 compatible volumes, 26 are chosen arbitrarily (the first 26 compatible volumes returned by FindFirstVolume as described in [MSDN-FNDFSTVOL] and FindNextVolume as described in[MSDN-FNDNXTVOL]).

A volume is considered compatible if all of the following are true:

- The volume is formatted with the NTFS file system.
- The file system supports ObjectIDs.

  This is determined using the GetVolumeInformation request, and checking in the *lpFileSystemFlags* parameter to GetVolumeInformation that the FILE_SUPPORTS_OBJECT_IDS flag is set. For more information on the GetVolumeInformation request, see [MSDN-GETVOLINFO].

- The volume is nonremovable.

  This is determined using the GetDriveType request, and checking that the return value is DRIVE_FIXED. For moreinformation on the GetDriveType request, see [MSDN-GETDRIVETYPE].

<14> Section 3.2.6.6: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: As new, compatible volumes become available on the local computer, they are added to the ClientVolumeTable.

<15> Section 3.2.6.6: All versions of Windows listed in the supported products list in Appendix B: Product Behavior: The VolumeInformation information is physically stored on the volume. So if a volume is reformatted such that there is no data on the volume, there will be no VolumeInformation table associated with that volume.

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

Q

R

S

T

V