IBM TotalStorage SAN File System
(based on IBM Storage Tank™ technology)

System Management API Guide and Reference

Version 2 Release 1

GA27-4315-01

IBM TotalStorage SAN File System
(based on IBM Storage Tank™ technology)

System Management API Guide and Reference

Version 2 Release 1

GA27-4315-01

Note
Before using this information and the product it supports, read the information in ″Notices.″

Second Edition (June 2004)

This edition applies to the IBM TotalStorage SAN File System and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office servicing your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for reader's comments is provided at the back of this publication. If the form has been removed, you may address your comments to:

International Business Machines Corporation
Design & Information Development
Department CGFA
PO Box 12195
Research Triangle Park, NC 27709–9990
U.S.A.

You can also submit comments by selecting Feedback at www.ibm.com/storage/support/.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

About this guide

This guide introduces the administrative agent for SAN File System. It describes the administrative agent object model, the classes and properties that make up the model, and methods that the classes provide to implement the model.

Who should use this guide

This guide should be used by application programmers writing third-party applications for SAN File System.

Note: This document is intended for selected business partners. Contact your IBM® representative before using this publication.

Application programmers should have experience in the following skills, or have access to personnel with experience in these skills:

- Object-oriented programming
- CIM-based application programming
- Networking and network management
- SAN management

Notices in this guide

The following types of notices occur in this guide and convey these specific meanings:

Note: These notices provide important tips, guidance, or advice.

Attention: These notices indicate possible damage to programs, devices, or data. An attention notice appears before the instruction or situation in which damage could occur.

Publications

This topic describes the publications in the SAN File System library and in related libraries.

SAN File System publications

This topic describes the publications in the SAN File System library.

The following publications are available in the SAN File System library. They are provided in softcopy on the *IBM TotalStorage SAN File System Publications CD* and at www.ibm.com/storage/support. To use the CD, insert it in the CD-ROM drive. If the CD does not launch automatically, follow the instructions on the CD label.

Note: The softcopy version of these publications are accessibility-enabled for the IBM Home Page Reader.

- *IBM TotalStorage SAN File System Release Notes*

  This document provides any changes that were not available at the time the publications were produced. This document is available only from the technical support Web site: www.ibm.com/storage/support

- *IBM TotalStorage SAN File System Software License Information*

  This publication provides multilingual information regarding the software license for IBM TotalStorage SAN File System Software.
- *IBM TotalStorage SAN File System Administrator's Guide and Reference*, GA27-4317

  This publication introduces the concept of SAN File System, and provides instructions for configuring, managing, and monitoring the system using the SAN File System console and administrative command-line interfaces. This book also contains a commands reference for tasks that can be performed at theadministrative command-line interface or the command window on the client machines..
- *IBM TotalStorage SAN File System Basic Configuration for a Quick Start*, GX27-4058

  The document walks you through basic SAN File System configuration and specific tasks that exercise basic SAN File System functions. It assumes that the physical configuration and software setup have already been completed.
- *IBM TotalStorage SAN File System Maintenance and Problem Determination Guide*, GA27-4318

  This publication provides instructions for adding and replacing hardware components, monitoring and troubleshooting the system, and resolving hardware and software problems.

  Note: This document is intended only for trained support personnel.
- *IBM TotalStorage SAN File System Installation and Configuration Guide*, GA27-4316

  This publication provides detailed procedures to set up and cable the hardware, install and upgrade the SAN File System software, perform the minimum required configuration, and migrate existing data.
- *IBM TotalStorage SAN File System Messages Reference*, GC30-4076

  This publication contains message description and resolution information for errors that can occur in theSAN File System software.
- *IBM TotalStorage SAN File System Planning Guide*, GA27-4344

  This publication provides detailed procedures to plan the installation and configuration of SAN File System.
- *IBM TotalStorage SAN File System System Management API Guide and Reference*, GA27-4315

  This publication contains guide and reference information for using the CIM Proxy API, including common and SAN File System-specific information.

  Note: This document contains information and procedures intended for only selected IBM Business Partners. Contact your IBM representative before using this publication.

SAN File System related publications

These publications are related to SAN File System.

- *IBM TotalStorage*® Subsystem Device Driver User's Guide, SC26-7637

Web sites

This topic discusses any Web sites that offer additional, up-to-date information about SAN File System.

The following Web sites have additional information about SAN File System:

- www.ibm.com/storage/support

- www.ibm.com/storage/software/virtualization/sfs

Chapter 1. Getting started

This chapter introduces the Common Information Model (CIM) agent for SAN File System, known as the *administrative agent*. The administrative agent implements an object-oriented management interface over Hypertext Transfer Protocol (HTTP). It conforms to CIM 2.7 and plans to follow the Storage Management Initiative Specification (SMI-S) as it develops. The administrative agent's managed object format (MOF) derives from CIM standard models where they apply to SAN File System.

This chapter describes key concepts related to CIM, SAN File System, and the administrative agent. It also presents functional views of the administrative-agent object model and describes programming considerations such as accessing the object model and invoking dynamic and static methods.

The rest of the information in this guide is organized as follows:

- Chapter 2, "Managing SAN File System," on page 61 describes how to perform tasks for managing SAN File System using the object model.
- Chapter 3, "Administrative agent methods," on page 75 describes intrinsic and extrinsic methods that the administrative classes provide.
- Chapter 4, "Administrative agent object classes," on page 87 describes the classes that make up the object model of the administrative agent and their properties and methods.
- The appendices provide the following additional information:
  - Accessibility features of the SAN File System console and help system
  - SNMP Trap MIB
  - Notices

CIM concepts

This section provides an overview of the Common Information Model (CIM) and CIM-related concepts. It also describes a CIM agent, in general, and the Storage Management Initiative Specification (SMI-S).

CIM

The Common Information Model (CIM) is a set of standards from the Distributed Management Task Force Inc. (DMTF). It provides a conceptual framework for storage management and an open approach to the design and implementation of storage systems, applications, databases, networks, and devices.

The CIM specifications provide the language and the methodology for describing management data. Specifically, the CIM defines common object classes, associations, and methods. Member vendors can use those objects and extend them to specify how data should be processed and organized in a specific managed environment.

CIM-related concepts

The CIM specifications use the following concepts and terminology to describe the various object models:

Namespace
The scope within which a CIM schema applies.

Schema
A group of object classes defined for and applicable to a single namespace. Within the CIM agent, the supported schemas are the ones that are loaded through the managed object format (MOF) compiler.

Managed object format (MOF)
A compiled language for defining classes and instances. A MOF compiler offers a textual means of adding data to the CIM Object Manager repository. MOF eliminates the need to write code, thus providing a simple and fast technique for modifying the CIM Object Manager repository (DMTF CIM Tutorial Glossary).

Object name
An object that consists of a namespace path and a model path. The namespace path provides access to the CIM implementation managed by the CIM agent, and the model path provides navigation within the implementation.

Class The definition of an object within a specific hierarchy. An object class can have properties and methods and serve as the target of an association.

Property
An attribute that is used to characterize instances of a class.

Key A property that is used to provide a unique identifier for an instance of a class. Key properties are marked with the Key qualifier. (DMTF CIM Tutorial Glossary).

Method
A way to implement a function on a class.

Qualifier
A value that provides additional information about a class, association, indication, method, method parameter, instance, property, or reference.

CIM agent

A CIM agent is a specific piece of software that handles CIM requests in an embedded model.

Components

A CIM agent typically contains (or interacts) with the following components:

The agent
An open-system standard that interprets CIM requests and responses as they are transferred between the client application and the device.

Client application
A storage management program that initiates CIM requests to the CIM agent for the device.

CIM object manager (CIMOM)
The common conceptual framework for data management that receives, validates, and authenticates the CIM requests from the client application and then directs the requests to the appropriate component or device provider.

Service location protocol (SLP)
A directory service that the client application calls to locate the CIMOM.

Device provider
A device-specific handler that serves as a plug-in for the CIM. That is, the CIMOM uses the handler to interface with the device.

Device
The storage server that processes and hosts the client application requests.

In the case of the CIM agent for SAN File System, known as the *administrative agent*, its components include the agent, the CIMOM, and the device provider. The administrative agent is separate from and interacts with the SLP daemon, client application and device.

CIM agent at work

The client application locates the CIMOM by calling an SLP directory service. When an application first invokes the CIMOM, it registers itself to the SLP and supplies its location information, including IP address, port number, and the type of service that it provides. With this information, the client application starts to communicate directly with the CIMOM.

The client application then sends CIM requests to the CIMOM. As requests arrive, the CIMOM validates and authenticates each request. It then directs the requests to the appropriate functional component of the CIMOM or a device provider. The provider makes calls to a device-unique programming interface on behalf of the CIMOM to satisfy client application requests.

Storage Management Initiative Specification

The Storage Management Initiative Specification (SMI-S) is a design specification of the Storage Management Initiative (SMI) launched by the Storage Networking Industry Association (SNIA).

It specifies a secure and reliable interface that allows storage management systems to identify, classify, monitor, and control physical and logical resources in a storage area network (SAN). The interface is intended as a solution that integrates the various devices to be managed in a SAN and the tools used to manage them. SMI-S is based on a number of existing technologies or industry standards that include the following:

Common Information Model (CIM)
An object model for data storage and management developed by the Distributed Management Task Force (DMTF). CIM makes it possible to organize devices and components of devices in an object-oriented pattern.

Web-Based Enterprise Management (WBEM)
A tiered enterprise management architecture also developed by the DMTF. This architecture provides the management design framework that consists of devices, device providers, the object manager, and the messaging protocol for the communication between client applications and the object manager. In the case of the CIM, the object manager is the CIMOM and the messaging protocol is the CIM over HTTP technology. The CIM over HTTP approach specifies that the CIM data is encoded in XML and sent in specific messages between the client applications and the CIMOM over the TCP/IP network in a SAN.

Service Location Protocol (SLP)
A directory service that the client application calls to locate the CIMOM.

Intended to be an industry standard, SMI-S extends the generic capabilities of the CIM, the WBEM, and the SLP to implement storage networking interoperability. For example, the WBEM is expanded to provide provisions for security, resource-locking management, event notification, and service discovery.

SAN File System concepts

This section discusses concepts that will help you understand how SAN File System works. Becoming familiar with the SAN File System components and understanding the concepts in this section enables you to use SAN File System most effectively.

Administrative server

The *administrative server* processes all requests that are initiated from an administrative interface. Three major components of the administrative infrastructure include IBM Director Agent, a Web server, and the administrative agent.

IBM Director Agent enables remote administration and control of the storage engines.

The Web server is the part of the administrative infrastructure that interacts with the administrative agent and renders the Web pages that make up the SAN File System console. The console is a Web-based user interface, which can be accessed using a Web browser, that has network access to the engines that host the master metadata server in the cluster.

An administrative server interacts with a metadata server through an intermediary service, called the *administrative agent*. The administrative agent is based on the Common Information Model (CIM) standard to process all management requests from the SAN File System console and administrative command-line interface. When you issue a request, the administrative agent checks with the lightweight directory access protocol (LDAP) server to authenticate the user ID and password and to verify whether the user has the authority (is assigned the appropriate role) to issue a particular request. After authenticating the user, the administrative agent interacts with the metadata server on behalf of that user to process the request. It also communicates with the operating system, the Remote Supervisory Adapter II (RSA) card, and administrative agents on other engines when processing requests. This same system of authentication and interaction is also available to third-party CIM clients to manage SAN File System.

To ensure high availability, the administrative server resides on each storage engine. All requests that come from the SAN File System console are always processed by the administrative server that runs on the same engine as the master metadata server. This server is known as the *primary administrative server*. However, requests that are initiated by the administrative command-line interface is processed by the administrative server that is running on the engine that you are logged in to. This can be the primary administrative server or a *secondary administrative server*, which is an administrative server that runs on an engine hosting a subordinate metadata server. All requests are redirected to a secondary administrative server if the primary administrative server is not available.

Alerts and events

An *event* is an occurrence in the metadata server or cluster, such as a change in state from online to offline. Events are recorded as messages in the metadata server logs. You can view these messages using the SAN File System console or by using administrative commands.

An *alert* is a message that can be generated for an event. It warns you about certain conditions, such as a fileset or a storage pool reaching or exceeding its threshold.

A *Simple Network Management Protocol (SNMP) trap* is a notification mechanism to convey the occurrence of an event. You can set configuration parameters that determine whether SNMP trap messages are generated for events. SNMP trap messages notify you about events asynchronously, eliminating the need for you to frequently view messages in the metadata server logs to determine the state of the cluster.

The first configuration parameter determines where SNMP trap messages are sent. An administrator specifies a list of SNMP managers that are the recipients of any SNMP trap messages. The list includes the IP address, port number, version of SNMP, and community string for one or two managers. If no SNMP managers are specified, no SNMP trap messages are sent.

The second parameter specifies which types of event messages also generate SNMP trap messages. Event message types are informational, warning, error, or severe. You can specify any combination of message severities.

In addition, if the call home feature is activated, that feature generates a specific type of SNMP trap whenever a metadata server encounters an event that requires notification to the IBM Support Center. When a metadata server encounters such an event, the following steps are taken by the call-home feature:

1. It sends an SNMP trap Protocol Data Unit (PDU) to the master console
2. It parses and converts the trap into a Simple Mail Transfer Protocol (SMTP) e-mail message.
3. The e-mail message is sent to a known SMTP mail server.
4. The e-mail message is forwarded to the IBM RETAIN® system for processing by support personnel.

All of the SNMP-related configuration parameters must be set correctly for the call-home feature to work.

Table 1. Alerts, events, and traps

	Description	Delivery
Alert	Warns of a significant event on a metadata server or cluster. Also informs about condition changes such as a change in state to offline, or storage is approaching capacity	A generated message is sent to the terminal
Event	Identifies an occurrence in the metadata server or cluster	The event is recorded in the metadata server logs.

Table 1. Alerts, events, and traps (continued)

	Description	Delivery
Trap	Optionally, notifies the administrator of events asynchronously	A generated message is sent to the administrator directly, either locally or remotely

Cluster

The SAN File System *cluster* is a set of metadata servers, each running on a separate hardware engine. The metadata servers communicate with each other and with SAN File System clients over your existing IP network. The cluster provides a single point of control for administrative and service operations.

The cluster has one master metadata server, which is initially designated by an administrator, and one or more subordinate metadata servers. The master metadata server maintains the cluster state and is the focal point for most administrative services. The maximum number of metadata servers that SAN File System allows in the cluster is eight.

Note: Although you cannot purchase SAN File System with only one metadata server, you can run a single metadata-server system if all other metadata servers in the cluster fail (for example, if you have only two engines, and one of them fails), or if you want to close all of the metadata servers except one to perform scheduled hardware maintenance.

Cluster workload

Each metadata server in a cluster, including the master metadata server, is assigned a *workload*; a workload is the metadata and locks that a metadata server is responsible for providing metadata to clients whenever they request access. A *fileset* is the unit of workload for metadata servers. Data resides in filesets assigned to metadata servers.

During client setup, a client is given the address of one of the metadata servers for initial contact and metadata server cluster discovery. When the client issues a request to access data, it is automatically directed to the appropriate metadata server to obtain the metadata and locks required to access the data.

SAN File System can automatically balance the workload for you, by dynamically assigning filesets across all metadata servers.

You can also choose to control the workload manually by statically assigning the filesets to specific metadata servers. When assigning static filesets, consider the metadata-activity level of each fileset, and balance the filesets evenly across all metadata servers in the cluster. To ensure that each metadata server handles a share of the entire workload, assign at least one fileset to each metadata server. Creating additional filesets provides you with greater flexibility in assigning and reassigning filesets to achieve optimal results. You can statically assign a fileset to a metadata server or unassign a fileset at any time.

A *spare metadata server* is an idle metadata server that has no statically assigned filesets. You can reserve a spare metadata server for failover to take on the workload of another metadata server that goes offline and preserve the workload balance.

Tip:

- Use all static or all dynamic fileset assignments in the cluster because in mixed environments, filesets could be reassigned to a metadata server that does not have spare capacity to handle the additional workload.
- The global fileset (ROOT) is created during installation time as a static fileset assigned to the master metadata server. You can reassign this fileset to another metadata server or change it to a dynamic fileset.

Communication between metadata servers

The metadata servers communicate with each other for a variety of reasons. For example, they exchange *heartbeats*, which are messages sent periodically from one metadata server to another so that each knows that the other is still active. If a metadata server stops sending heartbeats for a specific period of time (which is set with heartbeat parameters), the other metadata servers form the cluster again without it.

The master metadata server communicates with its subordinate metadata servers to perform many administrative tasks, such as supplying the metadata servers with the current workload map and querying metadata server status. It also contacts them to process requests from administrators to perform tasks such as draining volumes and creating FlashCopy® images of filesets.

Subordinate metadata servers initiate contact with the master metadata server for specific tasks, such as acquiring more space or obtaining file placement policy information.

Soft cluster failures

A *soft failure* by the cluster requires no intervention and is recoverable by simply restarting the metadata server software.

Soft failures are handled in different ways depending on their cause. The administrative server provides an optional metadata server restart service that monitors the metadata server software and restarts it if necessary. If a metadata server goes offline, an internal thread detects that condition and enables the rebooting of the software. If the operating system on the engine crashes or hangs, SAN File System can reboot the operating system. Then, the metadata server restart service is automatically started, and in turn, restarts the metadata server on the engine. When a metadata server is restarted after a soft failure, it resumes serving the same workload that it was serving before the failure.

If a subordinate metadata server fails, the filesets that are assigned only to that metadata server are temporarily unavailable. If the master metadata server fails, all of the subordinate metadata servers eventually pause, either because they require a service from the master metadata server or because they are no longer receiving a heartbeat from the master metadata server, and the entire global namespace is unavailable until the master metadata server is restarted.

When a client has an active session with a metadata server and that metadata server fails, the client stops receiving responses for transactions and lease renewal attempts. However, the client remains active. The client loses its lease and any locks it had obtained from that metadata server. After the metadata server is restarted, the client can contact the metadata server and renew its lease. Then, it can reassert its locks (get back all the locks it had before the metadata server failure) and refresh its metadata cache as needed. A metadata server provides a grace period for lock reassertion to allow clients to reassert their locks before allowing other clients to obtain new ones. Applications running on the client experience a pause in service during the restart and recovery period.

Components

The following figure illustrates the major components of SAN File System.

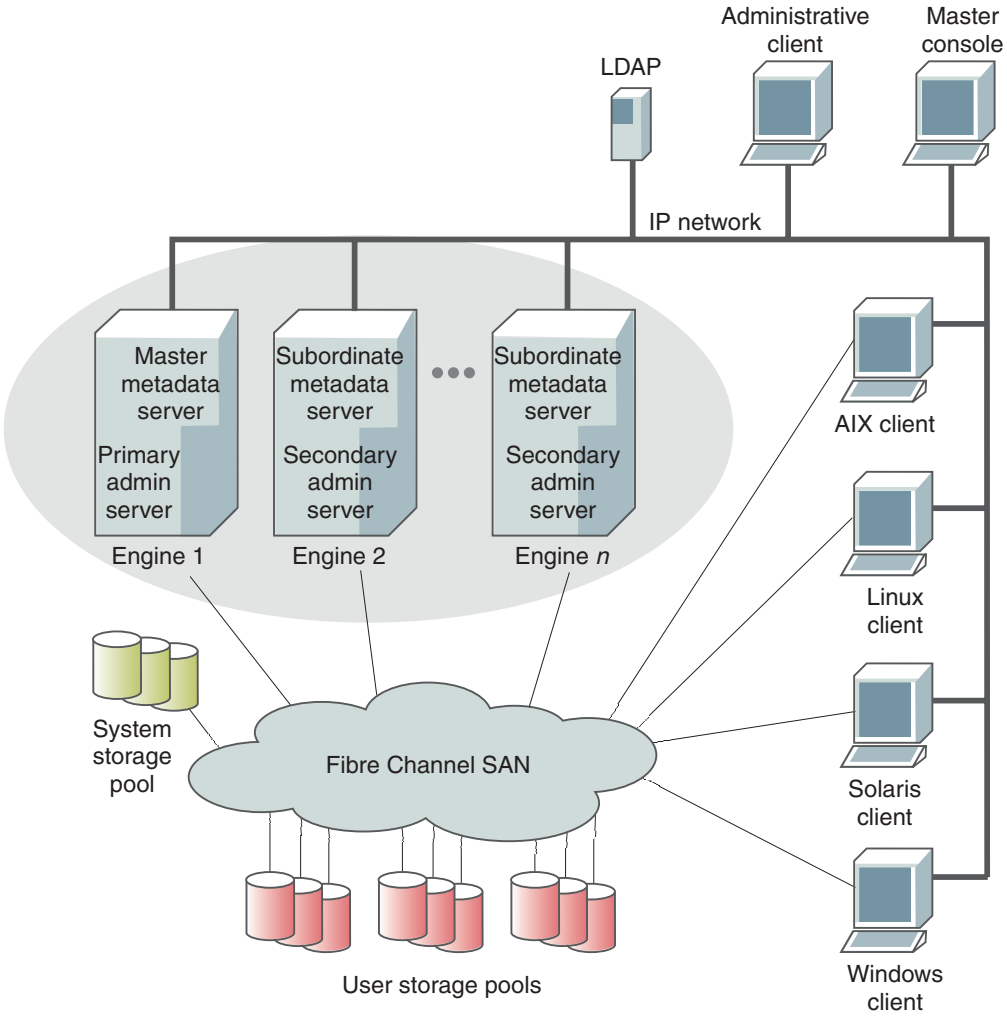


Figure 1. SAN File System components

The metadata servers and clients communicate over an private IP network and access data over a Fibre Channel storage attached network (SAN). SAN File System relies on networking hardware (including an IP network, SAN, network switches, and routers) that already exists in your environment.

The *metadata servers* run on separate physical machines (known as *engines*) and perform metadata, administrative, and storage-management services. The metadata servers are clustered for scalability and availability, and are referred to collectively as the *cluster*. In the cluster, there is one master metadata server and one or more subordinate metadata servers. Additional metadata servers can be added, as required, when the workload grows.

The metadata resides on private storage that is shared among all the metadata servers in the cluster. This storage is known as the *system storage pool*. A storage pool is a collection of SAN File System volumes in the SAN. The system storage pool contains the system metadata (such as system configuration and state

information) and file metadata (such as file creation date and permissions). The actual file data is stored on the *user storage pools*, which may be shared among the clients.

The *administrative server* allows SAN File System to be remotely monitored and controlled through a Web-based user interface, called the *SAN File System console*. In addition, the administrative server processes requests issued from the administrative command-line interface, which can also be accessed remotely. The ability to access the SAN File System through these two types of interfaces allows you to administer SAN File System from almost any system with network connectivity. The administrative server uses an *LDAP server* to look up authentication and authorization information about the administrative users. The primary administrative server runs on the same engine as the master metadata server. It receives all requests issued by administrators and also communicates with the administrative servers that run on each additional metadata server in the cluster to perform routine requests.

Computers that are going to share data and have their storage centrally managed are all connected to the SAN. In SAN File System, these computers are known as *clients*. The SAN File System client software enables the clients to access a single, uniform global namespace through a virtual or installable file system. These clients can act as servers to a broader clientele, providing network File System (NFS) or Common Internet File System (CIFS) access to the global namespace or hosting applications (such as database servers or Web-hosting services that use multiple servers).

The *master console* provides serviceability features, including the remote-support interface (or remote access) and service alert (for call home) capabilities. The master console is a required feature for SAN File System that can be shared with other IBM TotalStorage products, such as SAN Volume Controller.

Engines

Within SAN File System, the hardware on which a metadata server and an administrative server run is called a storage *engine*. SAN File System supports from two to eight engines.

SAN File System is intended to run with a minimum of engines. However, you can run a single-engine system if:

- All of the other engines fail (for example, if you have only two engines, and one of them fails)
- You want to bring down all of the engines except one before performing scheduled maintenance.
- One engine hosts a spare metadata server.

You can use the SAN File System console or administrative command-line interface to monitor and control the engines from any computer with a network connection to the cluster.

Filesets

In most file systems, a typical file hierarchy is represented as a series of folders or directories that form a tree-like structure. Each folder or directory could contain many other folders or directories, file objects, or other file-system objects, such as symbolic links or hard links. Every file system object has a name associated with it, and it is represented in the namespace as a node of the tree.

SAN File System introduces a new file system object, called a *fileset*. A fileset can be viewed as a portion of the tree-structured hierarchy (or global namespace). It is created to divide the global namespace into a logical, organized structure. Filesets attach to other directories in the hierarchy, ultimately attaching through the hierarchy to the root of the SAN File System cluster mount point. The collection of filesets and their content in SAN File System along with the file system root combine to form the global namespace. Fileset boundaries are not visible to the clients; only the administrator of SAN File System can see them.

From a client's perspective, a fileset appears as a regular directory or folder within which the clients can create their own regular directories and files. Clients cannot delete or rename the directories that represent filesets.

In addition to organizing the overall structure of the global namespace, SAN File System also uses filesets for these purposes:

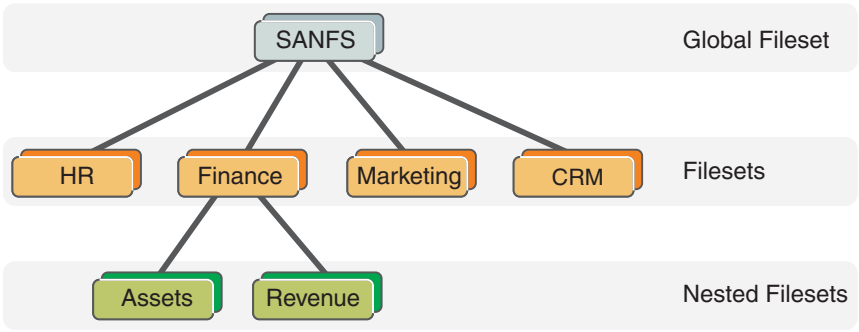
- Represent a unit of workload for the metadata servers
- Provide a level of granularity for data replication (using FlashCopy images)
- Control the amount of space used by the clients (through hard and soft quotas)

A fileset has the following properties:

- A fileset name.
- A directory path leading to the directory within which the fileset is attached. The directory path for the global fileset is the same as the cluster name, *sanfs*.
- A directory name that the fileset is given at the end of the directory path.
- A hard or soft quota.

The root of the global namespace is the *global fileset*. The name of the global fileset is always ROOT. The directory path of the global fileset is specified when you set up the global namespace and is the same as the cluster name *sanfs*.

When you create a fileset, you attach it to a specific location in the global namespace, either to the global fileset or to another fileset. When a fileset is attached to another fileset, it is called a *nested fileset*.



You can detach a fileset and reattach it at the same location or a different location. If a fileset is reattached at a different location, all the files contained in the fileset are rooted to the new location without any further operations. Before a fileset can be detached, any nested filesets must be detached first.

Filesets and clients

From a client perspective, a fileset appears to be a regular directory. Users and applications running on the clients can create objects, such as directories and files, within the fileset.

A fileset must be attached to the global namespace before it is available for use by clients.

Users cannot create hard links across fileset boundaries. In addition, a user cannot rename, move, or delete a directory that is the root of a fileset. If a user attempts to perform any of these operations, global namespace issues an error message.

Filesets and metadata servers

When creating a fileset, you may statically assign it to a specific metadata server or SAN File System can dynamically assign it to a metadata server for you. That metadata server is then responsible for providing metadata and locks to clients when they request access to files that reside in that fileset. The fileset-to-metadata server assignment is automatically communicated between clients and metadata servers. The client transparently discovers which metadata server to use when accessing files in a fileset. Each metadata server should be assigned to manage one or more filesets. If a metadata server is not managing any filesets, it is considered to be in standby mode. You can have an idle, or nearly idle, metadata server available to provide failover, if desired.

You should create at least one fileset for each metadata server in the cluster. However, creating more filesets gives you greater flexibility in distributing filesets among metadata servers in order to maintain availability and to balance the workload.

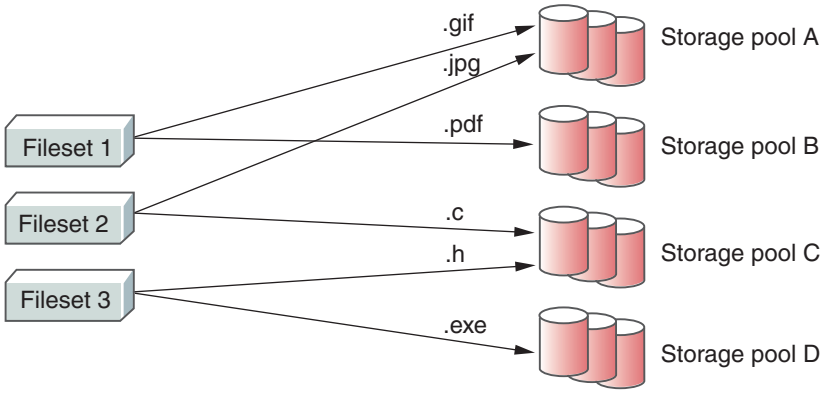
Tip: You can assign a nested fileset to a different metadata server than the one to which its parent fileset is assigned.

You can reassign a fileset to another metadata server, for example, to balance the workload. While filesets are being reassigned, they are temporarily unavailable to clients. After the reassignment, the clients can continue transparently and will automatically recognize the new metadata server hosting the fileset.

Filesets and storage pools

Filesets are not specifically related to storage pools, although each file in a fileset physically resides in blocks in a storage pool. This relationship is many-to-many; each file in the fileset can be stored in a different user storage pool. A storage pool can contain files from many filesets. However, all of the data for a particular file is wholly contained within one storage pool. The following illustration shows an

example of the relationship between filesets and storage pools.



When you create file placement policies, you can specify that all files created in a particular fileset are to be stored in a specific storage pool.

Fileset considerations

You can create filesets based on conditions in your environment (for example, workflow patterns, security, or backup considerations, all the files used by a specific application, or files associated with a specific application or client). Filesets are used not only for managing the storage space used, but also for creating FlashCopy images. Correctly defined filesets mean that you can take a FlashCopy image for all the files in a fileset together in a single operation, providing a consistent image for all of those files. The global namespace is partitioned into filesets that match the data-management model of the enterprise. Filesets can also be used as a criteria when placing individual files in global namespace.

Tip: When you are creating filesets, consider the overall I/O loads on the cluster. Because each fileset is assigned to one (and only one) metadata server, you need to balance the load across all metadata servers in the cluster by assigning filesets appropriately.

Separate filesets by their *primary allegiance* of the operating system to facilitate file sharing. Separating filesets also facilitates file-based backup methods (for example, utilities, such as tar, and Windows® backup applications such as VERITAS NetBackup or IBM Tivoli® Storage Manager); full metadata attributes of Windows files can be backed up from a Windows backup client only and full metadata attributes of UNIX® files can be backed up from an UNIX backup client only.

There are no known limitations to the number of filesets that you can create; however, when the number of filesets is greater than one thousand, response time will increase when you issue fileset commands.

Fileset permissions

When you create and attach a new fileset to the global namespace, the fileset is owned by user *Anonymous*. A UNIX root user or a Windows administrator user must change the ownership and permissions of the fileset before the fileset is usable. (You must do this for the FlashCopy directory and the lost+found directory under the fileset root.) You need to make these changes only once in the lifetime of a fileset. The changed permissions are persistent across metadata server restarts and whenever the fileset is detached or attached.

Unlike the requirement for the global fileset, a UNIX or Windows user can own a fileset exclusively. The fileset is not required to have write permissions for both UNIX and Windows domains.

Tip: If you change the permissions of a fileset after you create a FlashCopy image and then revert back to that FlashCopy image, the permissions also revert to their settings at the time when the FlashCopy image was taken.

Fileset quotas

When creating a fileset, you can specify a maximum size for the fileset, called a *quota*, and specify whether SAN File System should generate an alert if the size of the fileset reaches or exceeds a specified percentage of the maximum size, called a *threshold*. For example, if the quota on the fileset is set to 100 GB, and the threshold is 80%, and alert will be generated when the fileset contains 80 GB of data. (Note that the quota is based on space allocated to the fileset, not the data is contains.)

The action taken when the fileset reaches its quota size depends on whether the quota is defined as hard or soft. If a hard quota is used, once the threshold is reached, new client requests to add more space to the fileset (by creating or extending files) are denied. If a soft quota is used, which is the default, more space can be allocated but alerts continue to be sent. Once the amount of physical storage available to global fileset is exceeded, no more space can be used. The quota limit, threshold and quota type can be set individually for each fileset.

Note:

- The space used by a fileset includes the space used by FlashCopy images. It does not include the space used by any filesets nested within it.
- The metadata servers compute and track hard quota limits for filesets in multiples of the partition size. If a hard quota is not set as a multiple of the partition size, quota violation errors appear in the log file even though the size of the fileset has not reached the specified limit. To avoid this problem, specify hard quota limits as multiples of the partition size (for example, if the partition size is 16 MB, set the quota to multiples of 16).

Nested fileset considerations

Consider the following circumstances when creating nested filesets:

- You cannot access a nested fileset if the metadata server that is hosting the parent fileset is unavailable. In other words, if the metadata server hosting a fileset is offline, any nested filesets, even if hosted by a different metadata server, would also be offline.
- A FlashCopy image is created at the individual fileset level and does not include any nested filesets. You cannot make a FlashCopy image of a fileset and any nested filesets in a single operation. This can be of concern if you are required to have a consistent image of a fileset and its nested filesets. Making FlashCopy images in multiple operations could lead to ordering or consistency issues.
- To detach a fileset, you must first detach all of its nested filesets.
- It is not possible to revert to a FlashCopy image when nested filesets exist within the fileset. You must manually detach the nested filesets before reverting to the image. You can reattach the nested filesets after the fileset is reverted.
- When creating nested filesets, attach them only directly to other filesets. Do not attach filesets to client-created directories because a large-scale restore will be more complex.

FlashCopy images

The IBM TotalStorage SAN File System has a FlashCopy function, which creates an instantaneous copy or image of a fileset. The created image is a read-only, space-efficient image of the contents of a SAN File System fileset at the time that it was taken. You can use standard backup applications or utilities on SAN File

System clients to back up the contents of FlashCopy images, rather than the actual fileset. Backing up the image avoids any issues with open files that might cause problems when backing up live data.

FlashCopy images are file-based, so the SAN File System clients can see all of the files and directories in the image. The clients can use this image for quick restore of parts of the fileset if required, by simply copying the required files and folders back to the actual fileset. You can also quickly revert the entire fileset from a FlashCopy image.

Backing up and reverting files using FlashCopy images

When performing a backup using standard backup tools available in your environment, a client can specify the path to the FlashCopy image instead of the path to the actual files and continue working with the files while the backup occurs. This procedure produces a consistent backup of the files in the fileset.

Although creating FlashCopy images is *not* a replacement for creating backups of your files to protect your data, in some cases, a user can choose to use a FlashCopy image to revert a file or a set of files to a specific point in time. For example, if a user accidentally deletes a file, restoring it by copying it from a FlashCopy image to another directory instead of restoring it from a backup copy can be faster.

When you restore files from a backup taken from a FlashCopy image, you cannot restore the files to the same location – all FlashCopy image directories are read-only directories. You must restore the files to the directory where the original files resided or to another directory.

For ease of management, create FlashCopy images of all filesets at the same point in time, and use a common naming convention to indicate that they represent a set.

Copy on write

Immediately after the FlashCopy operation, the original fileset files (the source data) and the FlashCopy images (the copy data) of the files in the fileset share the same data blocks; that is, nothing is actually copied, which makes the operation space efficient.

As soon as any updates are made to the actual fileset contents (for example, a client adds or deletes files, or updates contents of files), the fileset is updated by an operation called *copy on write*; in which only the changed blocks in the fileset are written to a new location on disk. The FlashCopy image continues to point to the old blocks, whereas the actual fileset will be updated over time to point to the new blocks.

For example, when you create a FlashCopy image, the image is a set of pointers back to the original data in the fileset.

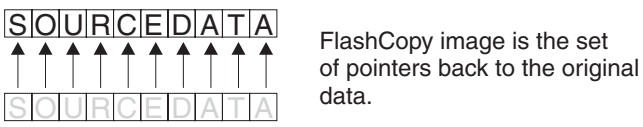


Figure 2. FlashCopy image

If two blocks were changed (S and E), one block was deleted (T) and a new block was written (P) in the actual fileset, the new blocks would be written, and the FlashCopy image continues to point to the original blocks, preserving the point-in-time copy.

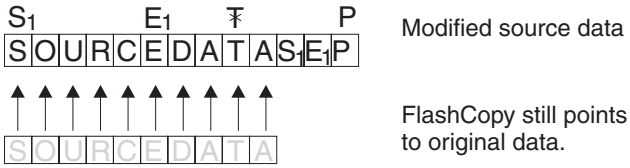


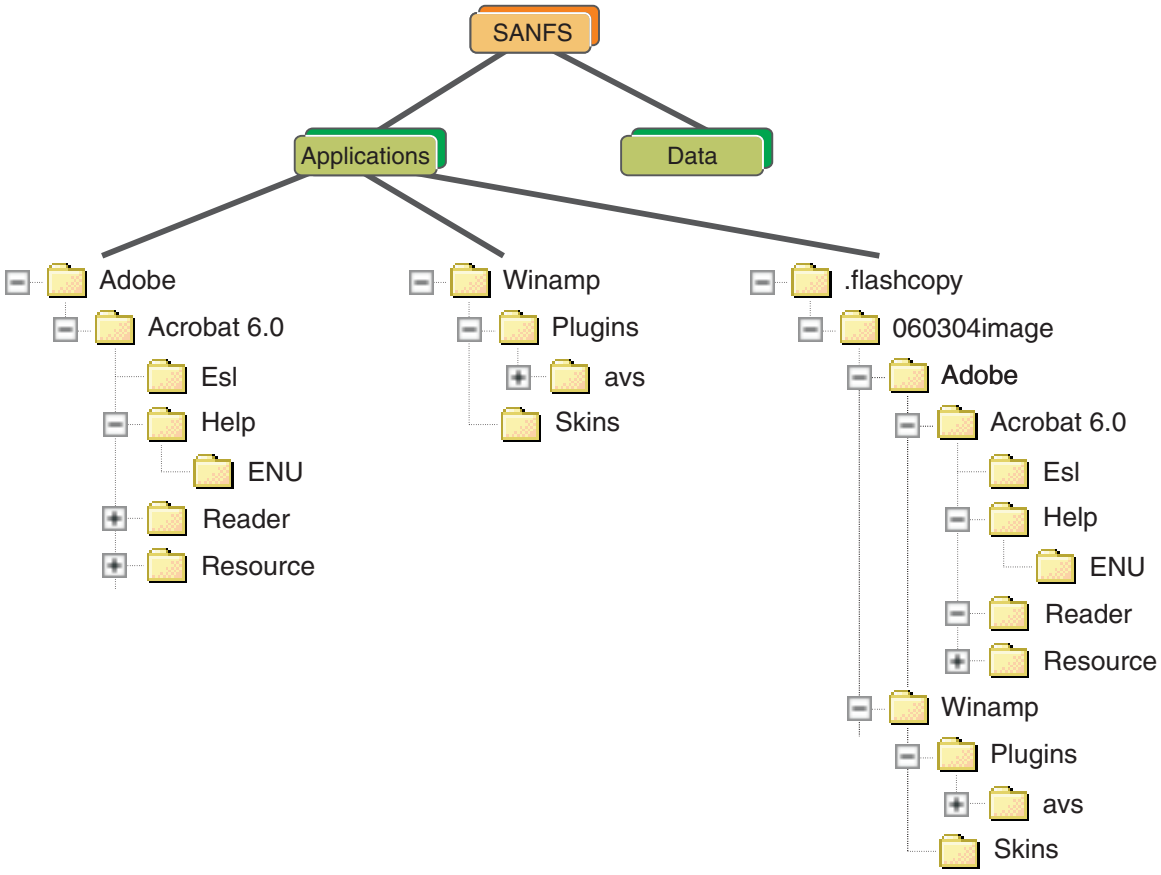
Figure 3. Copy on write

Therefore, any access to the FlashCopy image accesses the data blocks as they existed when the FlashCopy image was created, and any access to the fileset itself accesses the new data blocks.

Creating FlashCopy images

FlashCopy image is an image of an entire fileset as it exists at a specific instant. When you create a FlashCopy image, you specify the fileset to be copied. The FlashCopy image operation is performed individually for each fileset. While the FlashCopy image is being created, all data remains online and available to users and applications. The space used to keep the FlashCopy image is included in its overall fileset space; however, a space-efficient algorithm is used to minimize the space requirement. The FlashCopy image does not include any nested filesets within it. Also, you can create incremental FlashCopy images to be used as the basis for incremental backups. You can create and maintain a maximum of 32 FlashCopy images of any fileset.

The following figure shows how a FlashCopy image is viewable on a Windows client. In this case, a FlashCopy image was made of the Applications fileset and created in the directory 060304image. The fileset has two top-level directories, Adobe and Winamp. After the FlashCopy image is made, a subdirectory called 060304image appears in the special directory, .flashcopy (which is hidden by default), under the root of the fileset. This directory contains the same folders as the original fileset (Adobe, Winamp, and all the files and folder structure underneath). It is captured at the time the image was taken. Therefore, clients have file-level access to these images, and can access older versions of files, or make copies of individual files for the original fileset, if required.



When creating a FlashCopy image for a fileset, you can indicate whether the oldest image should be deleted if creating a new one causes the maximum number of images to be exceeded. Once a FlashCopy image is created, its name cannot be changed.

The actual files in a fileset and the FlashCopy images of the files in the fileset share the same file data blocks until a client makes changes to the files. When a client makes a change to a file, such as adding or deleting data, the client performs an operation called *copy on write*, in which the client writes the changed blocks to a new location on disk. At this point, the FlashCopy image points to the old blocks, and the actual file points to the blocks with the new data. Therefore, any access to the FlashCopy image produces the data blocks as they existed when the FlashCopy image was created, and any access to the actual file accesses the new data blocks.

FlashCopy image considerations

These are some basic considerations regarding FlashCopy:

- A FlashCopy image is simply an image of an entire fileset as it exists at a specific point in time.
- While a FlashCopy image is being created, all data remains online and available to users and applications.
- The FlashCopy image operation is performed individually for each fileset — that is, you can create only one FlashCopy image at a time.
- FlashCopy images are full images — you cannot create incremental FlashCopy images.
- Each fileset can have up to 32 read-only FlashCopy images.
- Once a FlashCopy image is created, its name cannot be changed.

- You can use a FlashCopy image for backing up files, instead of the original source data. This guarantees a consistent image of the files because the files in a FlashCopy image are read-only.
- Clients have file-level access to FlashCopy images, in order to access older versions of files, or to copy individual files back to the real fileset if required.
- FlashCopy images for each fileset are stored in a special subdirectory called .flashcopy under the fileset's attachment point. The .flashcopy directory is a hidden directory, so by default, it does not appear in Windows Explorer in a SAN File System client.

Disk space used by FlashCopy images

FlashCopy images consume space on the same volumes as the original fileset. Because FlashCopy uses a space-efficient method to make the image, the amount of space that is used by FlashCopy images is not possible to predict. If all blocks in the fileset are changed, the image takes up the same amount of space currently occupied by the non-FlashCopy objects within the fileset. If nothing in the fileset changes, the FlashCopy images takes up virtually no space (just pointers to the real fileset data). It is not possible to determine how much space is being occupied by a particular FlashCopy image at any particular time.

Therefore, when planning space requirements, include space for FlashCopy images. The amount of space you need to plan for flashcopy images correlates to the amount of changes you make to files with flashcopy images. Carefully monitor the user-storage-pool space threshold. Be aware that the space used by FlashCopy images count toward the fileset's quota.

Global namespace

The *global namespace* is the key to SAN File System. It allows common access to all files and directories to all SAN File System clients, and ensures that all SAN File System clients have consistent access and a consistent view of the data and files managed by SAN File System. Having common access to all files reduces the need to store and manage duplicate copies of data and simplifies the backup process. Security mechanisms, such as permissions and access control lists (ACLs), restrict visibility of files and directories.

Client access to the global namespace

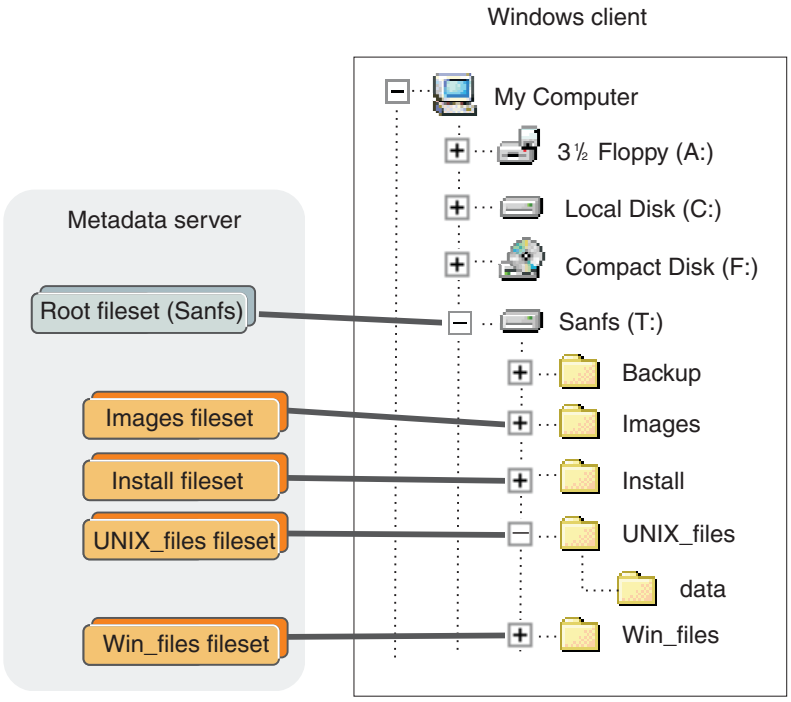
SAN File System clients mount the global namespace on their systems to access the filesets. After the global namespace is mounted on a client, users and applications can use it just as they do any other file system in order to access data and to create, update, and delete directories and files.

From a client's perspective, the global namespace appears as a normal directory. On a UNIX-based client, the global namespace looks like a mounted file system. On a Windows client, it appears as another drive letter and looks like any other NTFS file system. Basically, the global namespace looks and acts like any other file system on a client's system.

Note: A client cannot move, rename or delete a fileset, and cannot create hard links across fileset boundaries.

The following figure illustrates the appearance of the fileset from the metadata server and client perspectives. There are five filesets shown: the root, Images, Install, Unix_files, and Win_files. Some of these filesets have subdirectories (for example, the folder Backup is a subdirectory on the root filesystem, and the fileset

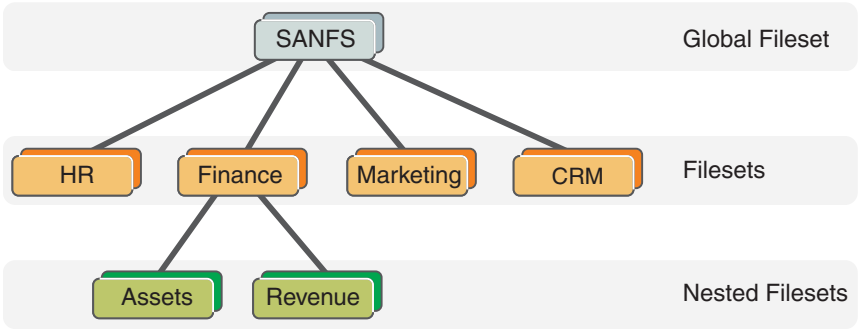
Unix_files, has a subdirectory named data). The client, however, cannot tell which folders are filesets; they appear all as regular directories.



Global namespace structure

The global namespace is organized into filesets. Each fileset is available to the global namespace at its attachment point. You are responsible for creating filesets and attaching them to directories in the global namespace.This can be done at multiple levels. An attach point appears to a SAN File System client as a directory in which the client can create files and directories (permissions permitting).

The following figure shows a sample global namespace. In this sample, the global fileset is attached to the root level in the namespace hierarchy (sanfs), and the filesets (HR, Finance, Marketing, and CRM) are attached to the global fileset, and the nested filesets (Assets and Revenue) are attached to the Finance fileset. By defining the path of a fileset's attach point, you also automatically define its nesting level in relationship to the other filesets.



Shared access to the global namespace

A *homogeneous environment* is one in which all clients run the same operating system. In a homogeneous environment, SAN File System provides access and

semantics that are customized for the operating system that is running on the clients. For example, when files are created and accessed from only Windows clients, all the security features of Windows are available and enforced. When files are created and accessed from only UNIX clients, all the security features of UNIX are available and enforced.

A *heterogeneous environment* is one in which clients run more than one type of operating system. In a heterogeneous environment, there is a restricted form of access. For example, when files created on an UNIX client are accessed by a Windows client, access is controlled using only the semantics and permissions of the "other" permission bits in UNIX. Similarly, when files created on a Windows client are accessed on an AIX® client, access is controlled using only the semantics and permissions of the "everyone" group in Windows.

Locks and leases

SAN File System uses locks and leases to ensure the consistency and integrity of data in the SAN File System global namespace. The internal locks discussed are in addition to the locks provided with the native file systems, such as flock() in UNIX.

A *lock* is a mechanism that restricts access to data and metadata. The SAN File System protocol provides locks that enable file sharing among SAN File System clients, and, when necessary, provides locks that allow clients to have exclusive access to files. It uses distributed data locks for cache consistency and file access locks to synchronize multiple, concurrent open instances of the same file. You can use locking semantics that correspond to open modes that are native to Windows and UNIX operating systems. You can also use byte-range locks.

When a client or server fails, SAN File System uses a lease-based safety protocol to ensure data consistency and to protect the structural integrity of the global namespace.

A client obtains a *lease* from a metadata server as soon as it makes contact with that server. A lease is valid for a period of time that is set by an administrator using a metadata server configuration parameter. When a client obtains a lock from a server, that lock is guaranteed to be valid by the server only as long as the client has a valid lease with the server. The server renews a client's lease each time the client contacts the server.

If a client does not contact the server within the specified lease period (for example, because of a temporary network failure), the server can revoke the client's locks. If other clients request locks on the same data, the server revokes the first client's locks and grants new locks to the other clients. If no such requests are made, when the client contacts the server again, it can renew its lease and reassert any locks (get its old locks back) that protect modified but uncommitted data in the client's cache, thus preventing data loss.

A client can also lose its lease because of a server failure. However, when the metadata server is restarted, the client can renew its lease and reassert its locks. A metadata server provides a grace period for lock reassertion to allow clients to reassert their locks before allowing other clients to obtain new ones. Reasserting the locks preserves the first client's cache. Clients cannot access any new data until the grace period has ended.

Logs and traces

SAN File System provides various logging and tracing mechanisms for use with the metadata server and clients. Logging messages provide a trail of routine system activities, operations that occur in normal day-to-day use of the product; as such, they are of interest to system administrators, as well as to IBM Support Center and trained service personnel. The contents of the logs, together with the observed symptoms, are used as a basis to begin isolating problems.

You can view the administrative, audit, cluster, and security logs from the SAN File System console or by using the **catlog** command; to clear the cluster and audit logs use the **clearlog** command. The SAN File System console also allows you to filter these logs based on severity and date.

Messages in the SAN File System console log views are linked to the Information Center. Clicking the message ID displays the Information Center with a description of the message and recommended actions to resolve the problem.

Administrative log

An *administrative log* contains entries for routine activity and error conditions that are generated by the administrative servers. A separate log is maintained for each administrative server. If you access these logs through the master metadata server (either from the SAN File System console or the administrative command-line interface, you see a consolidated view of the logs for each administrative servers in the cluster, ordered by date and time. If you access these logs through a subordinate metadata server (from the administrative command-line interface), you see only the logs for the administrative server running on the same engine as that particular metadata server.

Audit log

The *audit log* contains administrative audit messages, which are generated in response to operations performed by the SAN File System administrative server. The log does not capture every administrative operation, but records all commands that modify metadata or cluster configuration, or are significant operations, including commands that would have made such a change but failed to do so. The log also keeps a record of the user ID issuing the command, along with the time stamp and completion status of the requested operation. The log does not keep a record of simple query operations; such operations do not alter metadata, and because they are likely to be more numerous than those that do, their presence could easily overwhelm logging and interpretation of more meaningful operations.

The audit log is located on the engine that is hosting the master metadata server. When the master metadata server switches to another engine, the audit log on the original engine merges with the audit log on the new engine.

You can use information in the audit log to help convert requests made from the SAN File System console into equivalent command line interface instructions or to perform troubleshooting in the case of a failure within SAN File System.

Client logs and traces

Use the logging and tracing functions that are provided on each SAN File System client to help diagnose client-related problems. There is also a client dump facility to help with data collection in the event that IBM Support personnel requests this information.

Windows client logs, traces, and dumps

For Windows users, client log messages are written to the standard event log. To view the log, click **Start –> Programs –> Administrative Tools –> Event Viewer**.

Windows client tracing is written in the file C:\Program Files\IBM\Storage Tank\Client\log\sanfs.log for Windows. You must first enable tracing, using **stlog** command. You can view the trace log using a standard text editor.

You can configure Windows to generate a dump file if the Windows SAN File System client ends abnormally. By default, the file name is C:\WINNT\memory.dmp.

AIX client logs, traces, and dumps

In AIX, use the **stfsdebug** command and the **syslog** utility to enable tracing and logging on an AIX client. This client generates both log and trace messages, which are routed through the **syslog** utility on the AIX operating system. The **syslog** utility captures log and trace output from the kernel, as well as other operating system services. By default, the **syslog** utility discards all kernel output. However, you can configure the **syslog** utility to specify a destination for the messages by modifying the/etc/syslog.conf file. You use the **stfsdebug** command to enable tracing for an AIX client. In addition, you can specify which components (called classes) are traced, as well as the level of detail to include. You can also use **stfsdebug** to query the current status of all trace classes. Trace output enabled **stfsdebug** is sent to the syslog facility.

You can initiate a kernel dump of an AIX system if is still running but no longer responding to commands. Initiate a kernel dump using the **sysdumpstart** command from a system prompt using telnet or Secure Shell (SSH) tools; if you cannot establish remote access to the client, press the system reset button.

Linux client traces

To turn on tracing for the Linux client:

1. Verify that there is an entry in /etc/syslog.conf file for kern.debug (for example, kern.debug /var/log/kernmsg).
2. After the client module is loaded, edit the proc entries:
   a. Edit the "/proc/fs/stfs/debug" entry turn on tracing for the needed components (for example, change OFF to ON).
   b. Edit the "/proc/fs/stfs/csmdebuglevel" entry to set the CSM tracing level.

The trace output is placed in the location specified in the /etc/syslog.conf file (for example, /var/log/kernmsg).

Solaris client logs, traces, and dumps

You use the **sanfs_ctl** utility to enable tracing on a Solaris client. Logging and tracing output is stored in the standard system log in file /var/adm/messages.

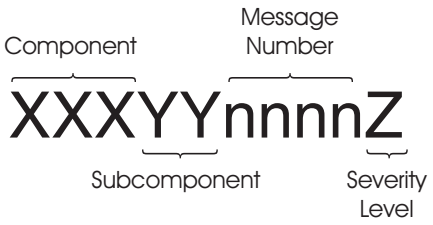
The Solaris operating system automatically creates a disk copy of the physical memory when a system crash occurs. This disk copy is known as a crash dump. The crash dump information is saved in a set of files named unix.*x* and vmcore.*x*, where *x* is an integer that identifies the dump. You can configure the directory in which these files and can set other dump parameters using the **dumpadm** utility.

Event log

The event logs contains a subset of the entries in the metadata server logs. The entries that appear in the event logs are those events, such as changes in server state, that have been configured as alerts. The event log is not stored in a separate physical file, but is generated from entries in the metadata server logs or cluster log.

Message IDs

The format of the message IDs helps you determine the type of error. The format of the message IDs is shown in the following figure. The*XXXX*, *YY*, and *Z* fields are alphabetic, and the *nnnn* field is a 4-digit number. For example, HSTAD001I would be an informational Basic Administration Message from the SAN File System Administration Service.



Refer to the "Message conventions" topic in the reference section for a detailed list of values for the component and subcomponent fields.

The severity can be one of these values:

I Informational
W Warning
E Error
S Severe

Metadata server logs

The *metadata server logs* contain entries for routine activity and error conditions that are generated by the metadata servers. A separate log is maintained for each metadata server in the cluster. If you access these logs through the master metadata server (either from the SAN File System console or the administrative command-line interface using the **catlog –log cluster** command), you see a consolidated view of the logs for each metadata server in the cluster, ordered by date and time, providing you with cluster-wide view of activities and events. This consolidated view is called the *cluster log*. If you access these logs through a subordinate metadata server (from the administrative command-line interface), you only see the logs for that particular metadata server.

Each metadata server log has a maximum size of 250 MB. Once a log file reaches its maximum size, it is renamed with a .old extension (for example, the log.std file is renamed log.std.old). An existing .old file is overwritten when a subsequent .old file of the same name is created. The log.std file is then cleared and used for new messages of the indicated type. In this way, 500 MB of each type of log data is maintained.

Security log

The *security log* maintains a history of administrator login activity generated by the administrative servers. A separate log is maintained for each administrative server. If you access these logs through the master metadata server (either from the SAN File System console or the administrative command-line interface, you see a

consolidated view of the logs for each administrative servers in the cluster, ordered by date and time. If you access these logs through a subordinate metadata server (from the administrative command-line interface), you see only the logs for the administrative server that is running on the same engine as that particular metadata server.

Trace log

The trace log receives trace messages from the metadata server. Because a minimal amount of tracing is always enabled for first-failure data capture, this log always exists. However, the number of messages and the level of detail that the messages convey is dependent on the current trace settings for the particular metadata server. The default level of tracing that is active at all times is 0, which sends only the most important messages. These messages are useful for providing initial first-failure data capture (FFDC) information. Trace messages are of interest primarily to IBM support personnel. Change the trace settings only at their direction. Higher levels of tracing can generate significant processing activity and should be used only when necessary.

Metadata server

A *metadata server* is a software server that performs metadata, administrative, and storage-management services and provides clients with shared, coherent access to shared storage (or global namespace). The metadata servers are clustered for scalability and availability, and are often referred to as a cluster. In the cluster, there is one master metadata server and one or more subordinate metadata servers, each running on a separate storage engine. Additional metadata servers can be added, as required, when the workload grows.

All of the metadata servers, including the master metadata server, share the workload of the global namespace. Each is responsible for providing metadata and locks to clients for specific filesets assigned to them. They know which filesets belong to which metadata server, and when contacted by a client, can direct the client to the appropriate metadata server. They also manage distributed locks to ensure the integrity of all of the data within the global namespace.

In addition to providing metadata to clients and managing locks, metadata servers perform a wide variety of other tasks. They process requests to create and manage filesets, storage pools, volumes, and policies; enforce the policies to place files in appropriate storage pools; and send alerts when any threshold established for the filesets and storage pools are exceeded.

Administrative services

The metadata servers process requests from administrators (issued from the SAN File System console or administrative command-line interface) to perform the following types of tasks:

- Create and manage filesets, which are subsets of the entire global namespace and serve as the units of workload assigned to specific metadata servers.
- Create and manage volumes, which are LUNs labeled for SAN File System's use in storage pools.

  Data-volume operations are initiated from and coordinated by the metadata servers but are actually performed by one or more clients. These operations are serial and I/O bound, and most have no effect on processing; however, some operations (such as volume drain) are more processor-intensive and might effect client performance. The metadata servers perform metadata-volume operations.
- Create and maintain storage pools. For example, an administrator can create a storage pool that consists of Redundant Array of Independent Disks (RAID) or

striped storage devices to meet reliability requirements, or create a storage pool that consists of random-access or low-latency storage devices to meet high performance requirements.
- Create FlashCopy images of filesets in the global namespace that can be used to make file-based backups easier to perform.
- Define policy sets that contain rules that determine in which storage pools specific files are stored.

Metadata services

There are two types of metadata:

- *File metadata* is information that clients need to access files directly from storage devices in the SAN. File metadata includes permissions, owner and group, access time, creation time, and other file characteristics, as well as the location of the file on the storage device.
- *System metadata* is metadata used by the system itself. It includes information about file sets, storage pools, volumes, and policies. The metadata servers perform the reads and writes required to create, distribute, and manage this information. The system metadata is stored and managed in the system storage pool, which is only accessible by the metadata servers in the cluster.

Distributing locks to clients involves the following services:

- Issuing leases that determine the length of time that a metadata server guarantees the locks that it grants to clients.
- Granting locks to clients that allow them shared or exclusive access to files or parts of files. These locks are semi-preemptible, which means that if a client does not contact the metadata server within the lease period, the metadata server can "steal" the client's locks and grant them to other clients if requested; otherwise, the client can reassert its locks (get its locks back) when it can contact the metadata server again.
- Providing a grace period during which a client can reassert its locks before other clients can obtain new locks if the metadata server itself goes offline and then comes back online.

Metadata servers and filesets

The metadata servers manage such things as file locations, file permissions, and locking. Only the metadata information travels over the IP network, minimizing the data transfer on the IP network. The SAN File System clients still access the regular data in the SAN and thus can benefit from the high performance a SAN can provide. For ideal performance, it is beneficial to evenly balance the workload across the metadata servers. Each fileset is assigned to an metadata server, and thus the filesets can be balanced across the metadata servers in the cluster.

SAN File System provides administrative commands that can be used to monitor transaction rates on each metadata server. Transaction rate parity across all metadata servers provides better SAN File System performance. Fileset assignment can be changed from one metadata server to another to balance transaction rates within the SAN File System environment. You should plan the filesets to be used based on expected I/O transaction rates, because this (rather than file size or storage space consumption) drives workload on the metadata server.

Storage-management services

The metadata servers perform these storage-management services:

- Manage allocation of blocks of space for files on LUNs
- Maintain pointers to the data blocks of the files

- Evaluate the rules in the active policy and manage the placement of files in specific storage pools based on those rules
- Issue alerts when file sets and storage pools reach or exceed their specified thresholds, or return out-of-space messages if they run out of space

SAN File System is designed to optimize communication between the metadata servers and clients. If a client deletes a file, there might be a delay before the free space is visible to the clients. The metadata server periodically checks for and reclaims freed space. Until the space is reclaimed, clients cannot see the freed space.

SNMP

The *Simple Network Management Protocol (SNMP)* is typically used to monitor network health, and performance and hardware, as well as to find and solve network problems. SNMP consists of two main components:

- SNMP agents, which are software components that reside on managed devices and collect management information (using Management Information Bases or MIBs). SNMP agents issue traps when SNMP events occur. These traps are sent through User Datagram Protocol (UDP) to an SNMP Manager.
- An SNMP manager, which is a network management application (for example, IBM Tivoli NetView®) that monitors and controls devices on which SNMP agents are running and can receive SNMP traps.

In SAN File System, each metadata server generates SNMP traps in response to certain events. SNMP traps are not issued from the operating system, hardware, or the administrative agent.

Tip: The RSA II cards can be set up to generate hardware traps as well.

You can configure which severity levels of events (informational, warning, error, or severe) should generate SNMP traps and you can define which SNMP managers in the SAN environment are to receive the traps. When an event occurs with a severity level that causes an SNMP trap, SAN File System sends the trap, and logs the event in the cluster log.

Note: SAN File System supports asynchronous monitoring through traps but does not support SNMP GETs or PUTs for active management. The SNMP Manager cannot manage SAN File System.

Not all events in SAN File System generate traps. Examples of events that might generate SNMP trap messages include:

- When a metadata server executes a change in state
- When a metadata server detects that another metadata server is not active
- When the size of a file set reaches a specified percentage of its capacity

Storage management

SAN File System provides automatic file placement through the use of policies and storage pools. You can create storage pools that are available to all clients, and define *rules* in policies that cause newly created files to be placed in the appropriate storage pools automatically. For more information about policies, rules, and storage pools, see the related topics below.

File placement

SAN File System provides automatic file placement at the time of creation through the use of policies and storage pools. You can create quality-of-service storage pools that are available to all users and define rules and policies that cause newly created files to be placed in the appropriate storage pool automatically.

A *policy* is a list of rules that determines where the data for specific files is stored. A *rule* is an SQL-like statement that tells a metadata server to place the data for a file in a specific storage pool if the file attribute that the rule specifies meets the specified criteria. A rule can apply to any file being created or to only files being created within a specific fileset depending on how it is defined.

The rules in a policy are processed in order until the condition in one of the rules is met. The data for the file is then stored in the specified storage pool. If none of the conditions specified in the rules of the policy is met, the data for the file is stored in the default storage pool.

Note:

- Rules in a policy are evaluated only when a file is being created. If you switch from one policy to another, the rules in the new policy apply only to newly created files. Activating a new policy does not change the storage pool assignments for existing files. Moving a file does not cause a policy to be applied. You can create multiple policies, but only one policy can be active at a time.
- After a file has been created, you can check its storage pool assignment using the **statfile** command from the administrative command-line interface.
- If you base your policies on user IDs, be aware of how the UNIX **tar** command restores files from backup. During the restore, a file is first created from backup with the user ID of the performer of the backup and is then changed to the user ID of the original creator of the file. With SAN File System, the policy is applied to the file at creation, so the policy applies to the user ID of the performer of the backup rather than to the user ID of the original file creator.

Policies and rules

This topic describes how SAN File System automates the placement of newly created files into storage pools using policies and rules.

Policies and the rules that they contain are used to automatically assign files to specific storage pools.

Policies

A *policy* is a set of rules that determine where specific files are placed based on the file's attributes. You can define any number of policies, but only one policy can be active at a time. If you switch from one policy to another or make changes to a policy, that action has no effect on existing files in the global namespace. The new or changed policy is effective only on newly created files in SAN File System. Moving a file does not cause the policy to be applied.

A policy can contain any number of rules. There is no limit to the size of a policy.

SAN File System performs error checking for policies in the following phases:

- When you creates a new policy, the master metadata server checks the basic syntax of all the rules in the policy.
- When you activate the policy, the master metadata server checks all references to filesets and storage pools. If a rule in the policy refers to a fileset or storage pool that does not exist, the policy is not activated and an error is returned.
- When a new file is created by a client, the rules in the active policy are evaluated in order. If an error is detected, the metadata server responsible for creating the file logs an error, skips all subsequent rules, and assigns the file to the default storage pool. If a default pool does not exist, the file is not created and the metadata server returns an error to the client application.

If your environment is set up in a non-uniform zone configuration (in which clients cannot access all volumes), you need to ensure that the rules in the active policy place files into volumes that are accessible to the clients that use them.

Tip: When SAN File System is first installed, a default policy is created and remains active until you create and activate a new one. The default policy assigns all files to the default storage pool. Although the default storage pool is created when SAN File System is first started, you must assign volumes to it before it can be used. If a user or application on a SAN File System client attempts to create new files that would be assigned to the default storage pool, and there are no volumes assigned to it, the user or application receives `No Space` errors.

Rules

A *rule* is an SQL-like statement that tells the metadata server to place the data for a file in a specific storage pool if the file meets specific criteria. A rule can apply to any file being created or only to files being created within a specific fileset or group of filesets.

Rules can specify any of these conditions, which when matched, causes that rule to be applied:
- Fileset
- Filename or extension
- Date and time when the file is created
- User ID and Group ID on UNIX clients

SAN File System evaluates rules in order, from top to bottom, as they appear in the active policy. The first rule that matches determines the file's placement. In other words, when a client creates a file, SAN File System scans the list of rules in the active policy to determine which rule applies to the file. When a rule applies to the file, SAN File System stops processing the rules and assigns the file to the appropriate storage pool. If no rule applies, the file is assigned to the default storage pool.

Tip:
- After a file has been created, you can check its storage-pool assignment from the administrative command-line interface using the **statfile** command.
- You can use the **statpolicy** command from the administrative command-line interface to view the statistics about the policy rules.
- If you base your policies on user IDs, be aware of the manner in which the UNIX **tar** command restores files from backup. During the restore, a file is first created from backup with the user ID of the performer of the backup

and is then changed to the user ID of the original creator of the file. With SAN File System, the policy is applied to the file at creation, so the policy applies to the user ID of the performer of the backup rather than to the user ID of the original file creator.

- During a restore or migration, a rule that uses the creation date as the placement criteria will assign a file based on the time of the restore or migration, not the original creation time, and a rule that uses a user ID or group ID as the placement criteria will assign a file based on the user ID or group ID of the restore or migration. Therefore, do not use creation time, user ID or group ID to place file.

For detailed information about creating policies and rules, see the related topics below.

Storage pools

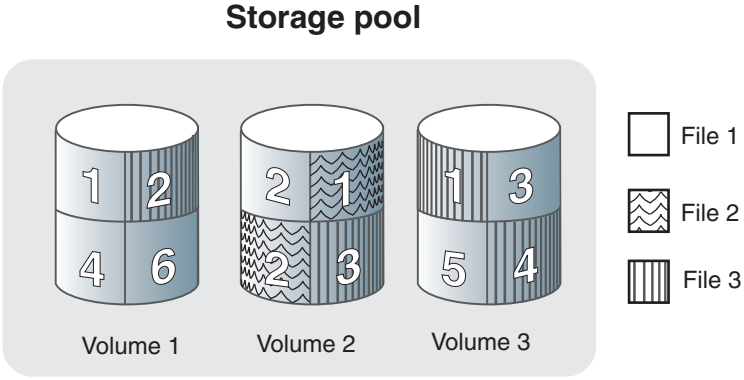
A *storage pool* is a named set of SAN File System volumes that can be used to store either metadata or file data. A storage pool consists of one or more volumes that provide a quality of service that you want for a specific use, such as to store all files for a particular application or a specific business division. You must assign one or more volumes to a storage pool before it can be used.

SAN File System has two types of storage pools: system storage pool and user storage pool.

Storage pools and volumes

Typically, you assigns volumes to storage pools based on their common characteristics, such as device capabilities (availability or performance level) and usage (business division, project, application, location, or customer).

Each storage pool manages its own volumes. File space is allocated to the volumes in a given storage pool in a round-robin algorithm (as shown in the following figure) in logical partitions, or in blocks. Logical partitions are allocated to the system storage pool in 16-MB blocks. For user storage pools, including the default storage pool, you can allocate logical partitions in 16, 64, or 256-MB blocks. All logical partitions in the same storage pool must be the same size.



Tip: You can set a threshold to generate an alert when a storage pool reaches or exceeds a certain percentage of its maximum capacity. By default, an alert is generated when a storage pool becomes 80% full. An alert is logged every five minutes until one or more volumes are assigned to the storage pool. You

can set configuration parameters to cause an SNMP trap message to be generated as well. An SNMP trap notifies you of this condition asynchronously.

System storage pool

The *system storage pool* contains the system metadata (system and file attributes, configuration information, and metadata server state) that is accessible to all metadata servers in the cluster. There is only *one* system storage pool that is created automatically when SAN File System is installed. The system storage pool contains the most critical data for SAN File System. The first volume that is assigned to the system storage pool, called the *master volume*, contains the most critical pages of metadata that SAN File System manages.

Important: Use highly-reliable and available logical unit numbers (LUNs) for the system storage pool (for example, mirroring or redundant array of independent disks (RAID), plus hot spares in the backend storage system) so that the cluster always has a robust copy of the system metadata.

Because the amount of metadata grows as the global namespace grows, you must monitor the system storage pool to ensure that there is always enough volumes assigned to it to accommodate the growth. The system storage pool typically requires approximately 2% to 5% of the total storage capacity that SAN File System manages, but this amount varies depending on your environment. Use the alert features on the system storage pool to ensure that you do not run out of space.

Tip: The minimum size of a system volume is 2 GB; therefore, the minimum size of the system storage pool is also 2 GB.

For security and reliability, the volumes that are assigned to the system storage pool should be accessible only to the cluster using a private SAN or a shared SAN with a combination of zoning, LUN masking, or special configuration. For reliability, the volumes should be virtualized RAID arrays (also known as *ranks* within IBM Enterprise Storage Server®).

User storage pools

A *user storage pool* contains the blocks of data that make up user files. SAN File System stores the data that describes the files, called file metadata, separately from the actual file data. You can create one or more user storage pools, and then create policies that contain rules that cause metadata servers to store data for specific files in the appropriate storage pools.

The *default storage pool* is a special user storage pool. This optional storage pool is used to store the data for a file if the file is not assigned to a specific storage pool by a rule in the active policy. A default storage pool is created when SAN File System is installed. However, if you want to use the default storage pool, you must assign one or more volumes to it. There can be only one default user storage pool in SAN File System. You can designate any user storage pool that has volumes assigned to it to be the default storage pool. You can choose to disable the default storage pool. In this case, newly created files that do not match any rules in the active policy are not saved.

User interfaces

There are two methods for managing SAN File System: an *administrative command-line interface* and a graphical user interface, called the *SAN File System console*. You can access the administrative command-line interface by either directly

logging in to an engine or using a Secure Shell (SSH) client to remotely connect to the engine. You can access the SAN File System console using a Web browser.

SAN File System provides you with different levels of user access to perform administrative operations. The users and user roles are defined on your LDAP server. Therefore, you cannot access the SAN File System without a valid user ID that is defined in the LDAP server.

SAN File System provides the following user interfaces:

- A Web-based administrative user interface called the SAN File System console
- An administrative command-line interface
- A client command-line interface

Note: The administrative server does not lock administrative access in order to prevent simultaneous SAN File System console or administrative command-line interface sessions. You must manually coordinate the use of the administrative interfaces.

SAN File System console

The SAN File System console allows you to control and monitor SAN File System from a Web-based graphical user interface. For ease of monitoring, it provides a system overview that illustrates the status of the various SAN File System components. In addition, the SAN File System console provides inline messaging that assists with system configuration, performance tuning and troubleshooting tasks.

The SAN File System console also contains the Help Assistant, which provides panel-level help information as well as links to related topics in the SAN File System Information Center. The Information Center serves as an online, searchable repository for all of the product documentation.

Administrative command-line interface

You can use the administrative command-line interface to administer all aspects of SAN File System, including setting up and managing storage pools, volumes, and filesets. For security reasons, administrative command-line interface runs only on the engines in your cluster.

You can use the administrative command-line interface interactively using the **sfscli** utility. You can also embed administrative commands in scripts.

To access **sfscli**, you must log in to an engine that hosts any metadata server. The following figure illustrates how you access **sfscli**.

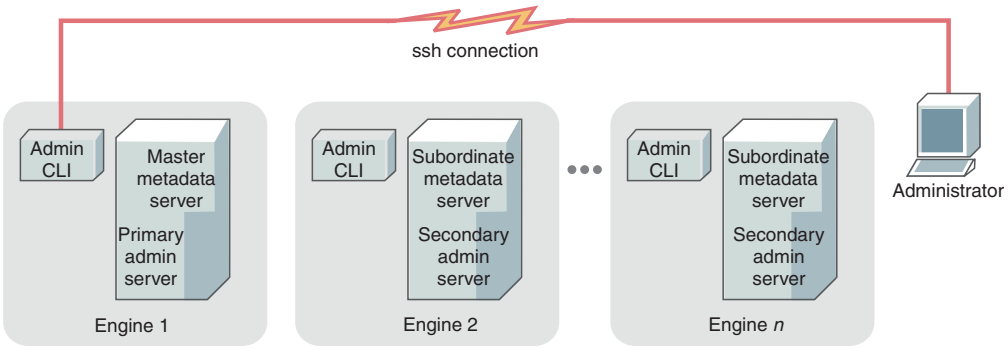


Figure 4. Accessing **sfscli**

Client commands

SAN File System provides a set of commands that are used to set up SAN File System clients and to perform planning, migration, and verification tasks for data. These commands are issued from the client machines.

User roles

SAN File System provides different levels of user access that you can assign to administrative tasks in your environment. These access levels, or *user roles*, are one way to provide security. The following table describes the SAN File System user roles.

Table 2. SAN File System user roles

Role	Level	Description
Monitor	Basic level of access	Allows you to obtain basic status information about the cluster, display the cluster message log, display the rules in a policy set, and list information regarding SAN File System elements such as storage pools, volumes, and filesets.
Backup	Monitor + backup access	Allows you to perform backup and recovery tasks in addition to all operations available to the Monitor role.
Operator	Backup + additional access	Allows you perform day-to-day operations and tasks requiring frequent modifications, in addition to all operations available to the Backup and Monitor roles.
Administrator	Full access	Provides you with full, unrestricted access to all administrative operations.

At least one user with Administrator access is required. You can also choose to define other roles as appropriate for your organization.

Volumes

A *logical unit number* (LUN) is the logical unit of storage that a SAN or other disk subsystem can assign to metadata servers and clients. A *volume* is a LUN that is labeled by SAN File System for its use. Volumes are grouped together virtually to form storage pools, in which file data and metadata is stored.

An LUN becomes a SAN File System volume when you add it to a storage pool. It is automatically assigned a system-generated label that identifies it as a SAN File System volume. You must also give the volume a name that is unique among all the volumes used by a SAN File System cluster.

During startup, the metadata server scans all LUNs that it can access in the SAN, searching for the label that tells it that the LUN is a valid SAN File System volume. Clients perform this same search whenever they are started.

System-data LUN operations are performed by the metadata servers. All other data LUN operations are initiated from and coordinated by the metadata servers in the cluster but are actually performed by one or more clients; therefore, the metadata servers no longer need to see the data LUNs, and the clients only need to see the data LUNs that they need to access. This allows SAN File System to support a wide variety of SAN configurations, storage devices, and drivers, and also supports scaling to large numbers of storage devices and clients. This also allows SAN File System to support grouping clients and LUNs into SAN zones to provide enhanced security.

A volume must be empty to be removed from a storage pool. When you remove a volume, SAN File System moves (drains) the contents of that volume across other available volumes in the same storage pool. If the storage pool does not have sufficient space available in other volumes to move all of the data contained in the specified volume, the removal fails and the metadata server suspends the volume (the metadata server cannot allocate new data on that volume).

Tip:

- Keep the storage subsystem device driver's virtual path (vpath) configuration file current. If many LUNs are added and deleted from the metadata server, it is possible for the configuration file to contain references to LUNs that do not exist.
- When the number of entries in the storage subsystem device driver's vpath configuration file reaches 255, any new LUN configured on the metadata server will not be visible.

Volumes and storage pools

When you install SAN File System, there is a system storage pool, which is used by metadata servers to store system and file metadata, and a default storage pool, which can be used to store file data. You can create additional user storage pools for file data; however, no data can be stored in a storage pool until you assign one or more volumes to it. You can also remove the default storage pool if you choose.

The volumes added to the system storage pool are called *system volumes*.

As the amount of metadata that is generated for the server cluster and client files grows, you must ensure that the system storage pool always has enough volumes assigned to it so that it does not run out of space.

You must also ensure that any user storage pools, including the default storage pool, has a sufficient number of volumes. Each storage pool must have at least one volume assigned to it before any files can be stored in it.

To assist you in monitoring storage pool capacity, SAN File System provides a threshold option that you can specify when adding a volume to a storage pool or changing settings for a storage pool. A threshold is a specified percentage of the estimated maximum capacity of the storage pool. When a storage pool reaches or exceeds the percentage specified as its threshold, SAN File System generates an alert. This alert can also generate an SNMP trap message to notify you of the condition asynchronously, if you set the appropriate parameters for SNMP traps.

Volume activation and suspension

When you add a volume to a storage pool, by default, the volume is activated. This means that a metadata server can allocate data to the newly added volume. You can add a volume to a storage pool in a suspended stat; however, no data can be allocated to the volume until you activate it.

A volume can be in a suspended state if you add it to a storage pool without activating it. You can also change the state of a volume from activated to suspended. When a volume is in a suspended state, a metadata server cannot allocate any data to it.

Volume removal

You can remove a volume from a storage pool. During this process, any files that are stored in the volume are automatically redistributed among the remaining volumes in the same storage pool. When you remove a volume, data is moved and committed one logical partition at a time. If a failure occurs while moving the contents of a volume, you can reissue the command, and the move process continues where it stopped earlier.

If I/O errors occur for some files or if the redistribution of the files to other volumes fails, you can specify a force option that causes the volume to be removed anyway. If there is a bad file on the volume, specifying this option causes the system to delete the entire file, even if parts of the file reside on other volumes.

If the bad file is part of a FlashCopy image, SAN File System removes it from the FlashCopy image. This removal might render the FlashCopy image non-revertible. You can still backup and restore the remaining files on the FlashCopy image.

Remember: There is no automatic recovery process when you specify the force option. All files are removed immediately without being copied. Before removing a volume with the force option, use the **reportvolfiles** command to display a list of files on the volume. The files for which failures occur are also listed in the cluster message log, and you can restore those files manually.

When you remove a volume from a storage pool, its label is removed, and the volume becomes a logical unit number (LUN) again.

Limitations to volumes in the system storage pool

The volumes in the system storage pool have these limitations:

- All volumes in the system storage pool must be of the same type of backend storage device and must be one of the supported IBM storage subsystems. You can use IBM TotalStorage SAN Volume Controller to provide mixed storage as long as only the SAN Volume Controller virtual devices are visible to the cluster.

- All volumes in the system storage pool must visible to all metadata servers in the cluster.
- Each volume in the system storage pool must be at least 2 GB in size.
- The system storage pool is limited to 126 dual-path volumes.

Administrative agent for SAN File System

The CIM Agent for SAN File System, known as the administrative agent, provides an application programming interface for the operations that an administrator performs to manage a cluster. It offers CIM-compatible objects for managing SAN File System.

Note: The SAN File System CIM model is following the direction of the Distributed Management Task Force Inc. (DMTF) industry standard as it develops. As the standard develops, the SAN File System model should improve in its consistency and its inheritance from CIM base classes. Also, the SAN File System model does not currently include associations and indications.

Service Location Protocol (SLP) is a mechanism for publishing and locating the administrative agent. It enables third-party clients to discover and connect to the administrative agent. SAN File System provides a default SLP configuration. If you want to modify the default configuration, see the www.openslp.org web site for information about SLP settings.

A standard CIM client can use the administrative agent to access and control the metadata server. The administrative agent restricts access to administrative operations through user roles that are stored in the Lightweight Directory Access Protocol (LDAP) server. Before using a CIM client, make sure the LDAP server contains your user name and password. See the *SAN File System Installation and Configuration Guide* for information about configuring LDAP.

Functional view of the Administrative agent

This section provides functional views of the administrative agent object model. Diagrams show specific functionality that is provided by the administrative agent and illustrate the architecture of the administrative agent.

CIM base classes

The following diagram shows the CIM base classes, which are the superclasses of the SAN File System classes.

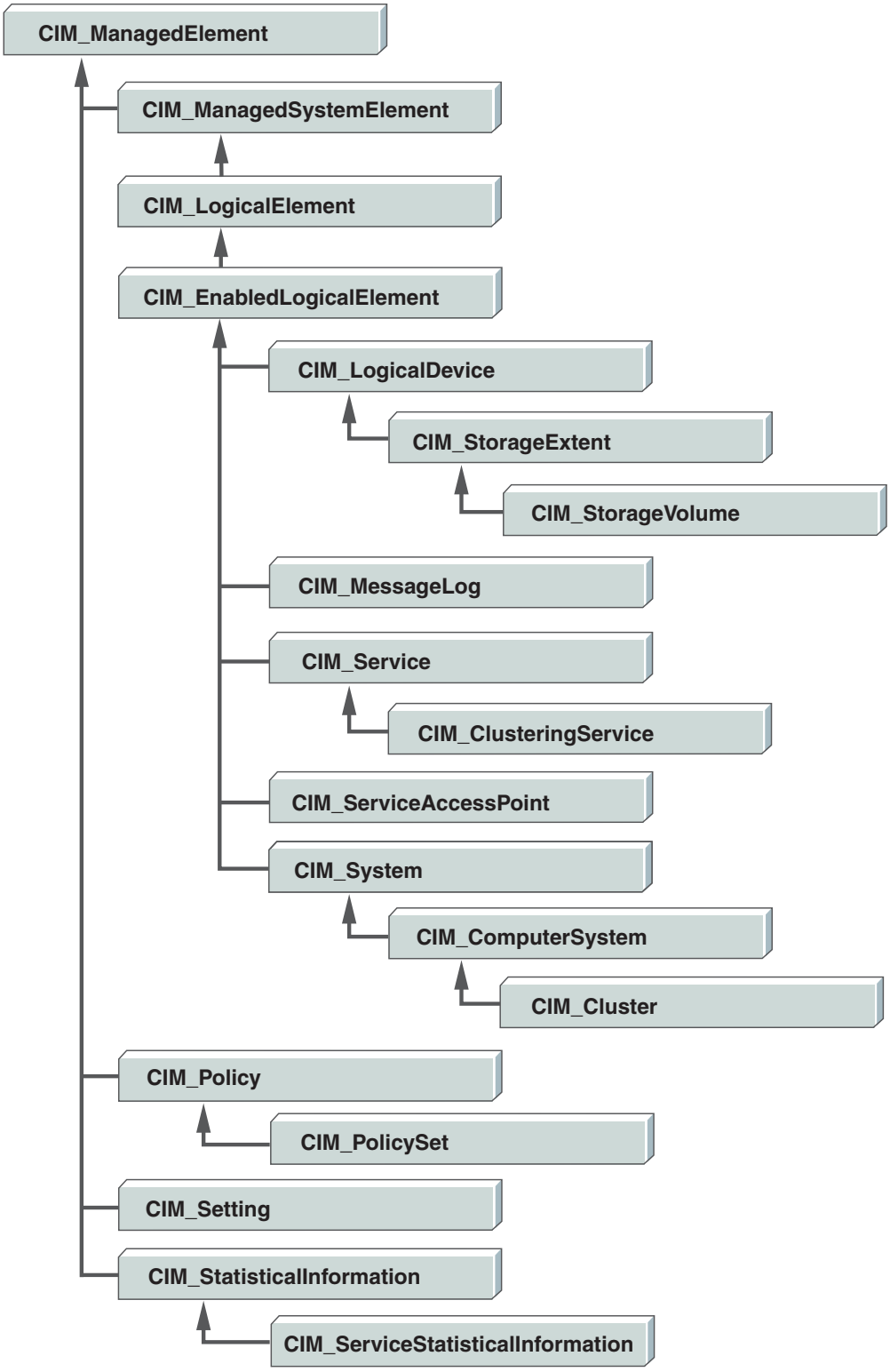


Figure 5. CIM base classes

SAN File System component classes

Table 3 provides an overview of the classes that represent the major elements of SAN File System.

Table 3. SAN File System element classes

Name	Description
"STC_AvailableLUNs" on page 89	This class represents an available Fibre Channel (FC) LUN. (When you assign a LUN to a storage pool, SAN File System labels it as a volume.) This class provides information about the channel and the LUN size and state. Its methods enable you to retrieve information about LUNs that a specific client can access.
"STC_Cluster" on page 94	This class, along with the STC_MasterService class, represents a cluster. It provides the identifier of the cluster and the number of engines.
"STC_ComputerSystem" on page 94	This class represents each engine in the cluster. It provides the identifier and state of the engine. Its methods enable you to set and retrieve the power state of the engine.
"STC_Container" on page 97	This class represents a fileset (also known as a container). It provides the identifier, its location, size, FlashCopy images and server of the fileset. Its methods enable you to define a new fileset, attach, detach, delete or move an existing fileset or change its hosting server. It also provides methods to remove the static assignment that a fileset has to a server, to list the storage pools that a fileset can use to store data, and to retrieve fileset information.
"STC_MasterSAP" on page 111	This class represents the master metadata server service access point.
"STC_MasterService" on page 111	This class, along with the STC_Cluster class, represents a cluster and provides cluster services. It identifies the service and the state of the cluster. Its methods enable you to bring up, bring down, quiesce or resume all servers in a cluster, add or drop a server from the cluster, start or stop a check of metadata, display metadata for a file, or commit the cluster to start using the latest software. It also provides methods for listing clients that have access to a specified LUN or volume.
"STC_PolicySet" on page 138	This class represents a policy, which is a list of file-placement and service-class rules that define characteristics and placement of files. It provides information about the policy and its policy rules. Its methods enable you to create, delete, activate, and retrieve the rules and statistics for a policy.
"STC_RegisteredFSClients" on page 143	This class represents a registered client of a metadata server. It provides identifier, location, and lease information about the client.
"STC_RemoteServiceAccessPoint" on page 144	This class represents a remote service access point. It provides information that you can use to access the SAN File System console.

Table 3. SAN File System element classes (continued)

Name	Description
"STC_StoragePool" on page 145	This class represents a storage pool. It provides identifier, type, and size information about the storage pool. Its methods enable you to create, delete, move, set the storage pool type, and disable the default storage pool.
"STC_TankSAP" on page 156	This class represents a metadata server service access point. It provides the addresses of the Ethernet server and the configuration of the ports and indicates whether the local server is the master metadata server.
"STC_TankService" on page 158	This class represents a metadata server and provides server services. It provides the identifier and state of the server, whether it is a master metadata server, and the number of filesets that it serves. Its methods enable you to start and stop a metadata server. It also provides a method that enables you to make a server the master metadata server during a rolling upgrade.
"STC_Volume" on page 164	This class represents a volume. It provides identifier, location, state and size information about the volume. Its methods enable you to create, delete, move, resize a volume and suspend and resume partition allocation of a volume, reset the iterator that locates each file entry on the volume, and return the next file entry.

Available LUNs class

The following diagram shows the hierarchy and definition of the STC_AvailableLUNs class.

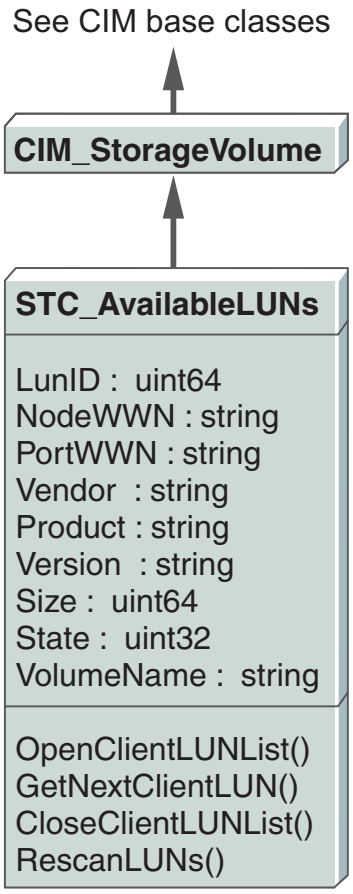


Figure 6. Available LUNs class

Computer system classes

The following diagram shows the hierarchy and definitions of the STC_Cluster and STC_Computer classes.

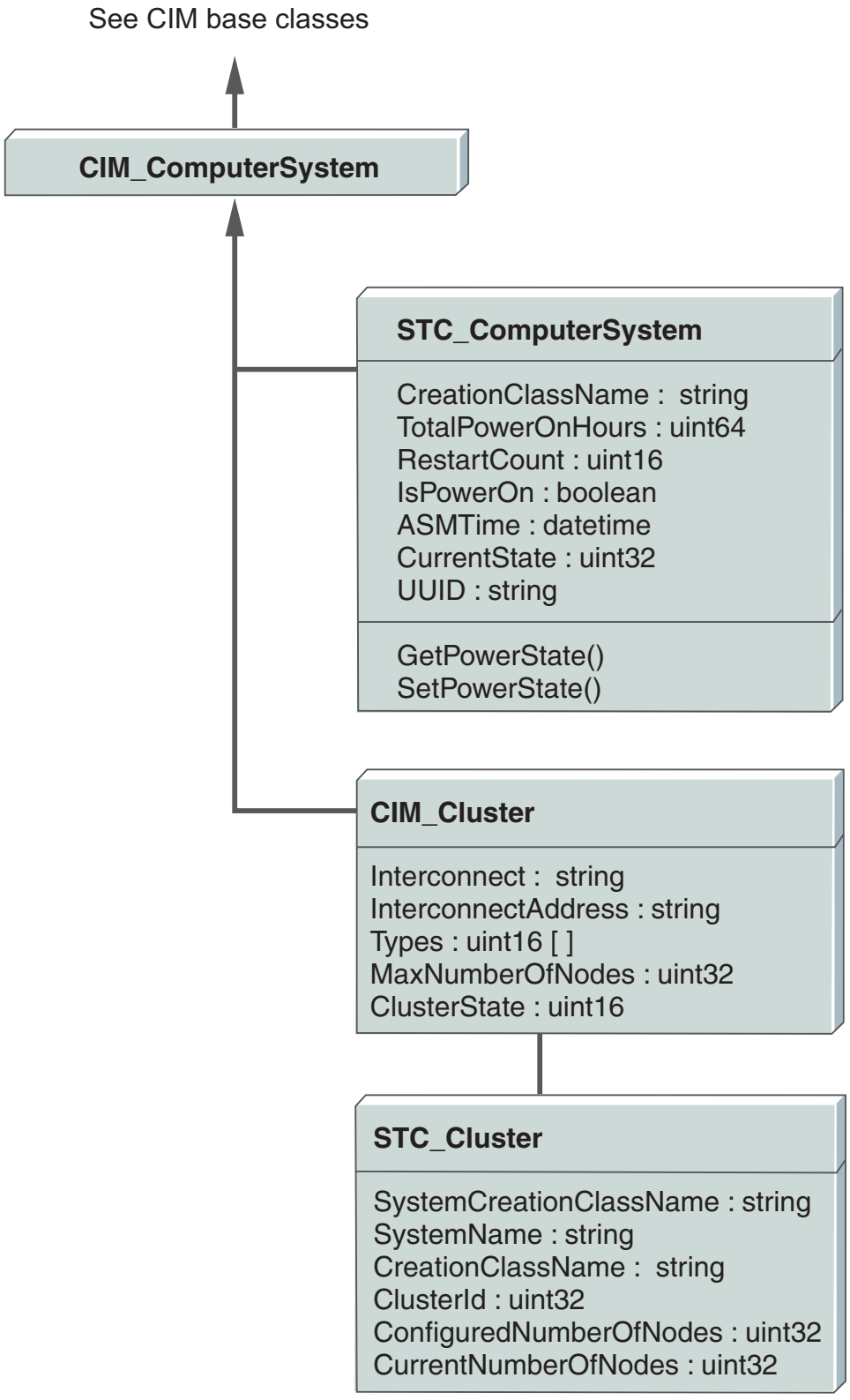


Figure 7. Computer system classes

System element classes

The following diagram shows the hierarchy and definitions of the STC_Container, STC_StoragePool, and STC_Volume classes.

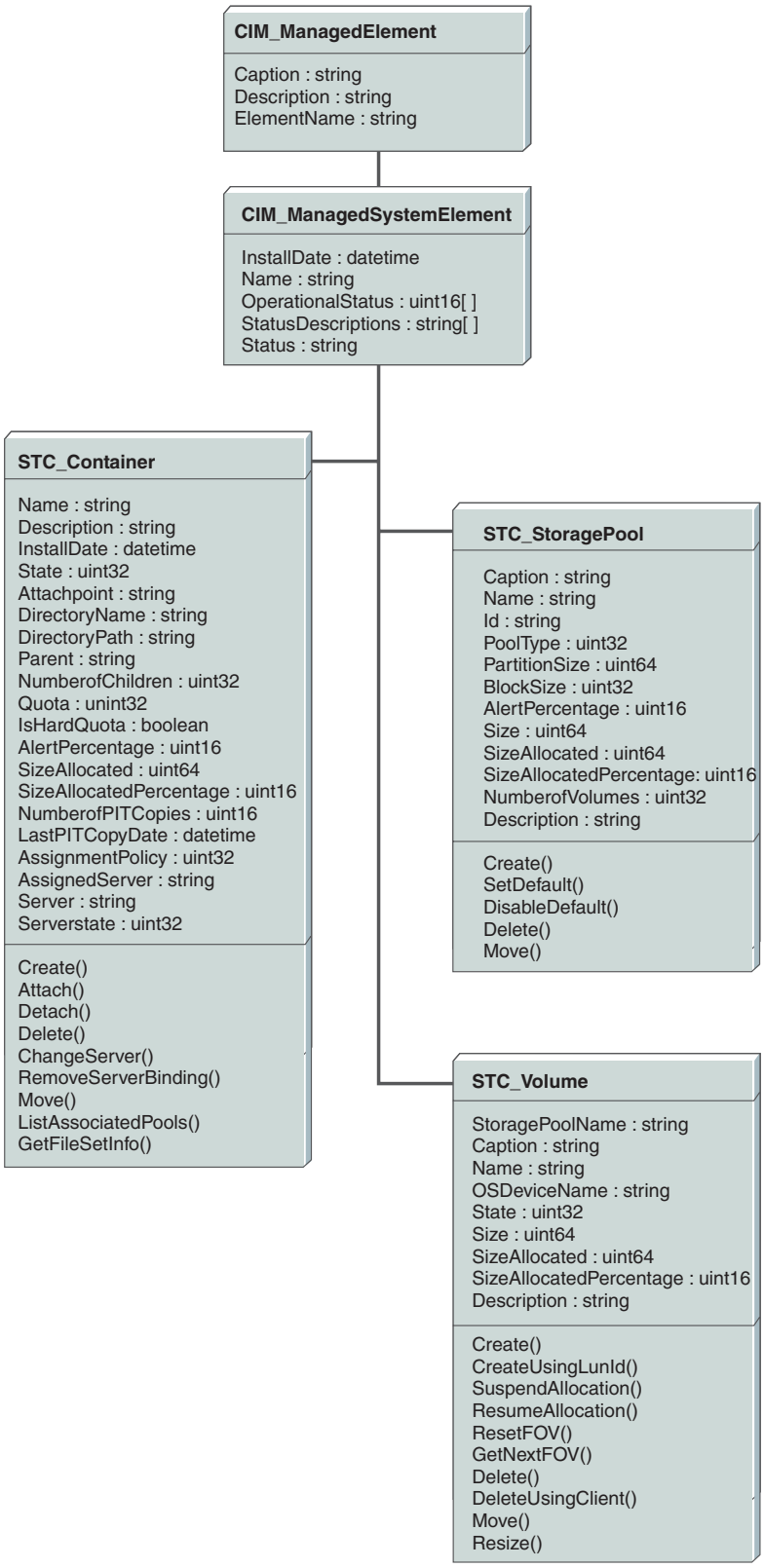


Figure 8. System element classes

Service access point classes

The following diagram shows the hierarchy and definitions of the STC_RemoteServiceAccessPoint, STC_MasterSAP, and STC_TankSAP classes.

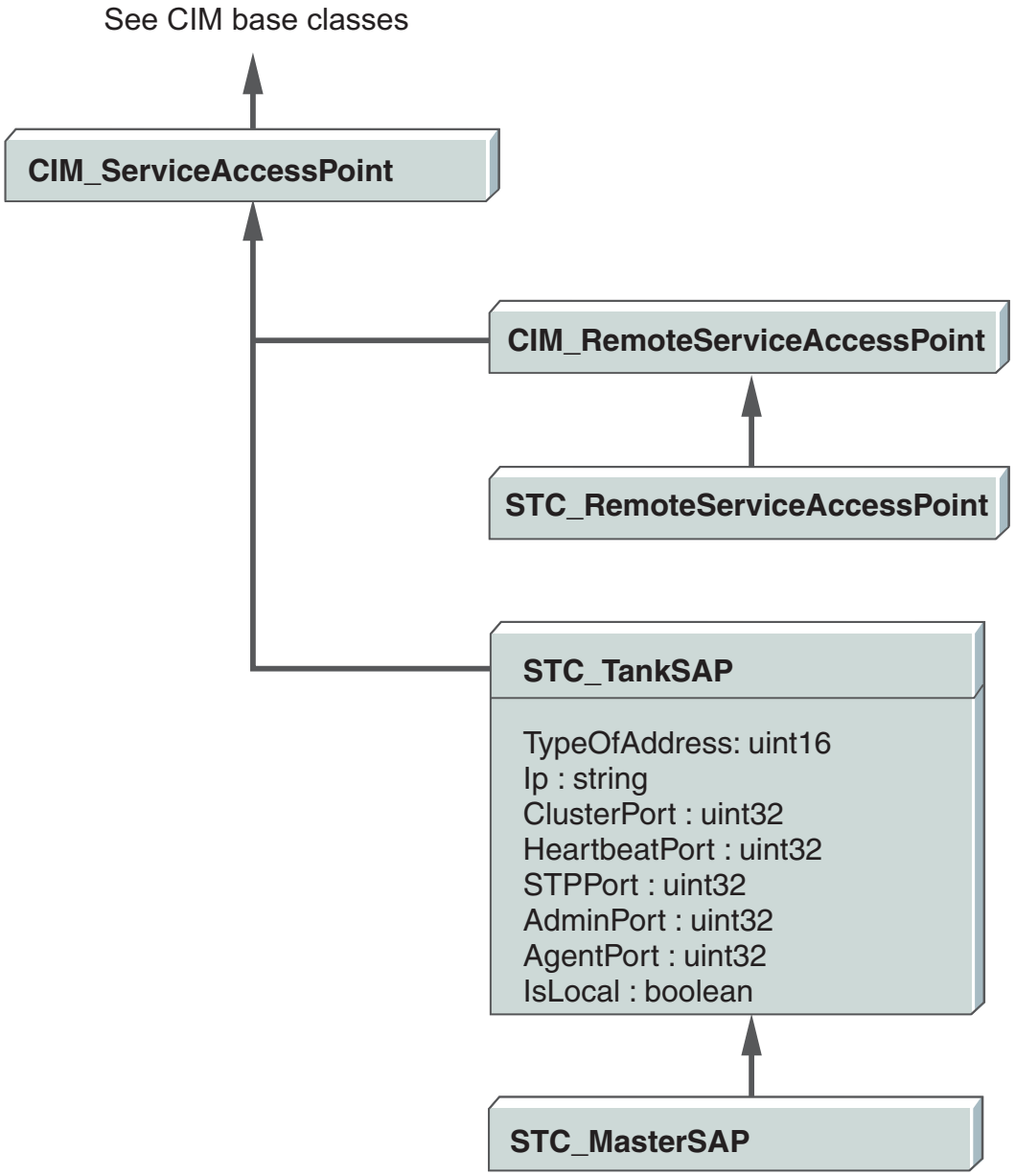


Figure 9. Service access point classes

Service classes

The following diagram shows the hierarchy and definitions of the STC_MasterService and STC_TankService classes.

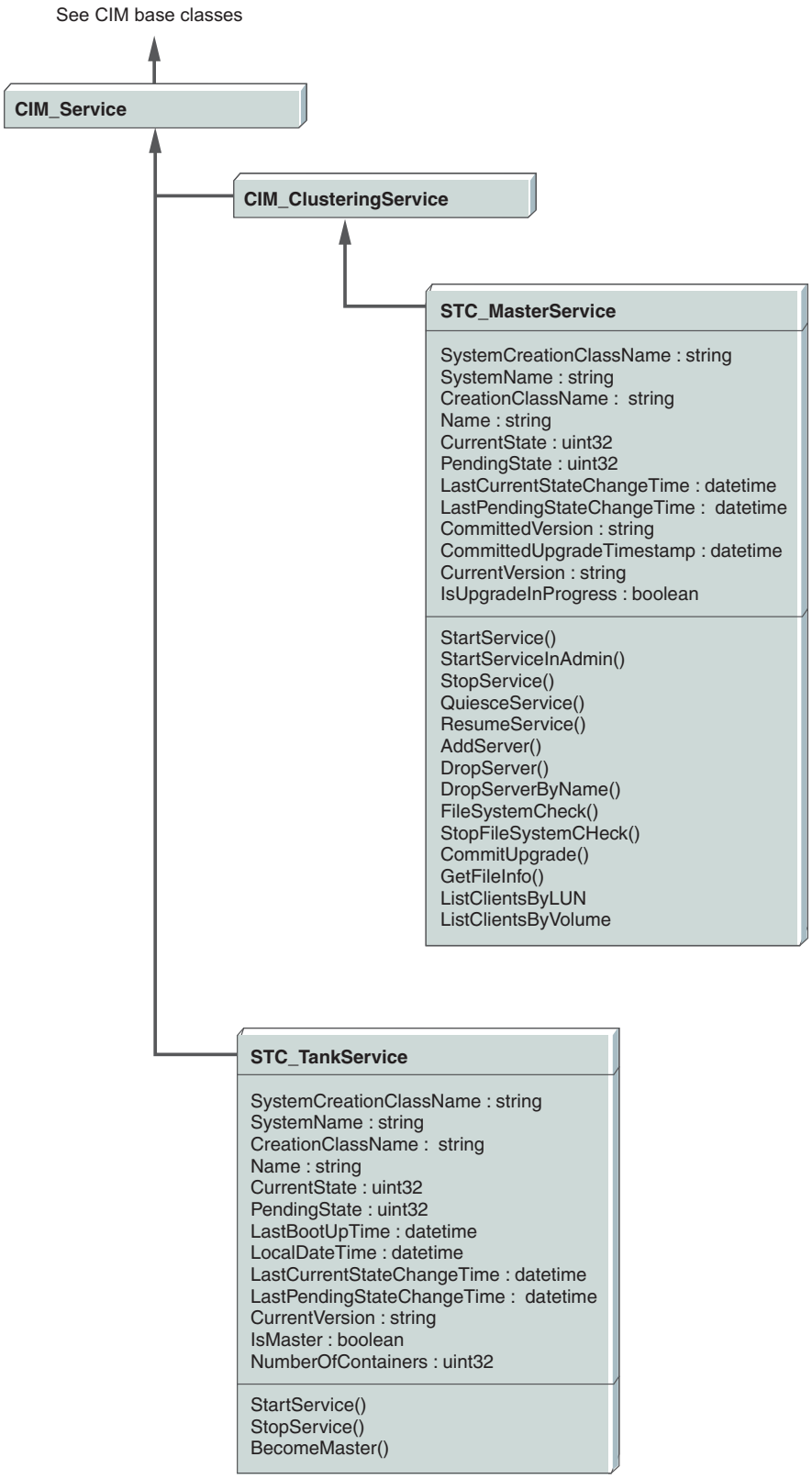


Figure 10. Service classes

Policy class

The following diagram shows the hierarchy and definition of the STC_PolicySet class.

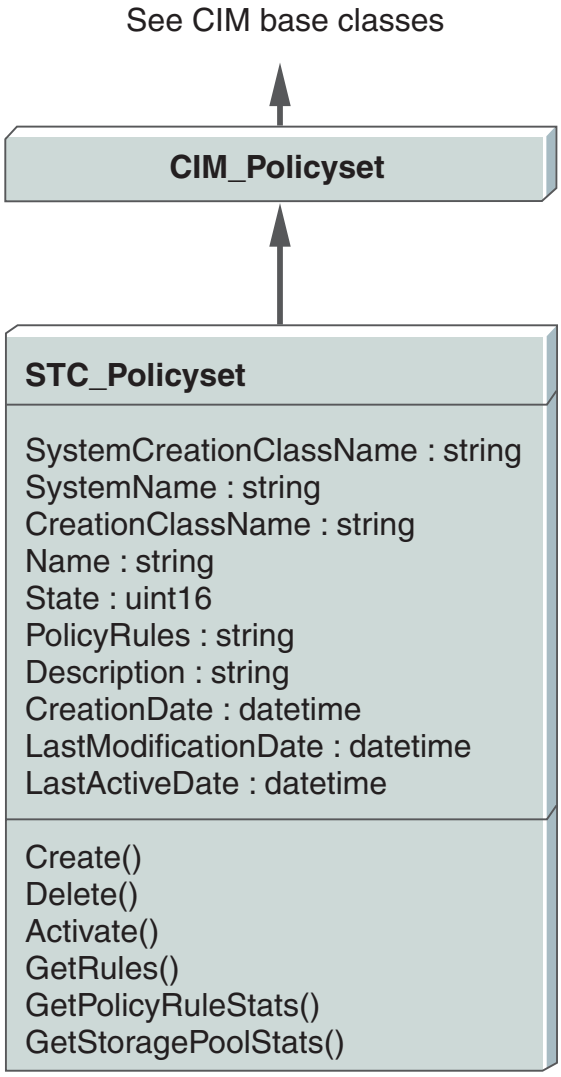


Figure 11. Policy class

Metadata server clients class

The following diagram shows the hierarchy and definition of the STC_RegisteredFSClients class.

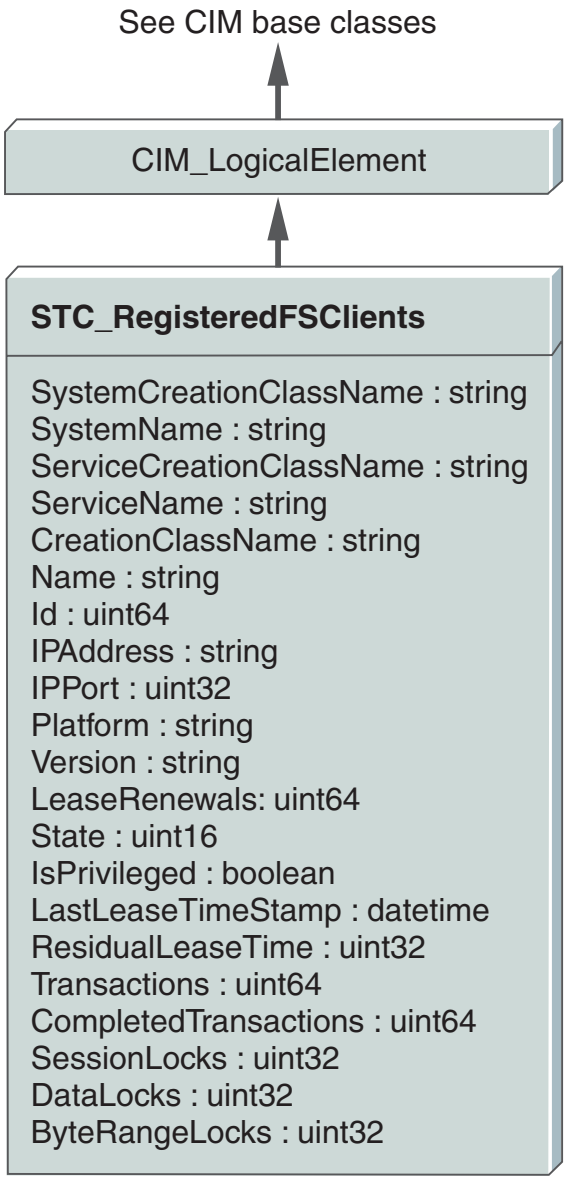


Figure 12. Metadata server clients class

SAN File System configuration classes

Table 4 provides an overview of the classes that represent configuration parameters.

Table 4. SAN File System configuration parameter classes

Name	Description
"STC_LdapDynamicSetting" on page 107	This class represents LDAP configuration parameters that you can dynamically update, without a cluster restart.

Table 4. SAN File System configuration parameter classes (continued)

Name	Description
"STC_MasterDisruptiveSetting" on page 107	This class represents cluster configuration parameters that require a cluster restart for an update to take effect. Its method enables you to list the languages installed in SAN File System.
"STC_MasterDynamicSetting" on page 109	This class represents cluster configuration parameters that you can dynamically update, without a cluster restart.
"STC_RsaDynamicSetting" on page 144	This class represents Remote Service Adapter (RSA) configuration parameters that you can dynamically update, without a cluster restart.
"STC_TankDisruptiveSetting" on page 152	This class represents settings for server-specific, configuration parameters that need a metadata server restart for an update to take effect.
"STC_TankTransientSetting" on page 160	This class represents server-specific configuration parameters that are effective only until the next restart.
"STC_WatchdogDynamicSetting" on page 174	This class represents metadata server restart service configuration parameters that you can dynamically update, without a cluster restart.

The following diagram shows the hierarchy and definitions of the configuration classes.

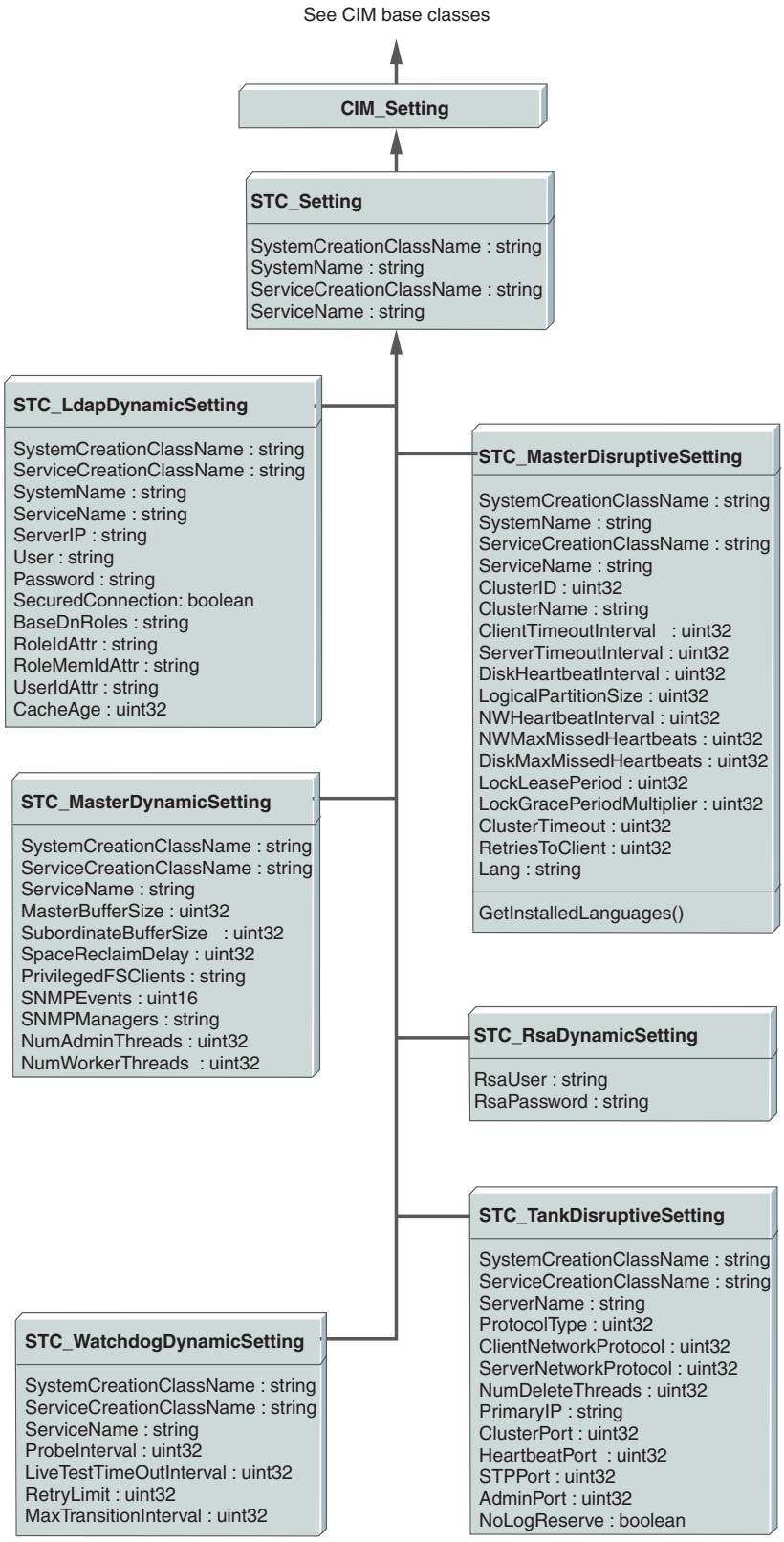


Figure 13. Configuration classes

SAN File System status classes

Table 5 provides an overview of the classes that represent the status of SAN File System components.

Table 5. SAN File System status classes

Name	Description
"STC_AdminProcess" on page 87	This class represents a long-running administrative process in the cluster of servers. It provides the identifier and start time for the process, and the command that initiated the process.
"STC_AdminUser" on page 88	This class represents an authorized user of the SAN File System. It provides a user's identifier and role as defined in the Lightweight Directory Access Protocol (LDAP). It also indicates whether the LDAP needs to reauthorize a user for a new request or if the user is still authorized from the last request, and the time remaining in that authorization window. Its methods enable you to clear this authorization window for an individual or all users.
"STC_MasterMetrics" on page 110	This class represents the metrics for a cluster. It provides metrics for metadata activity and buffers within the cluster.
"STC_NodeFan" on page 130	This class represents the status of a engine's fan. It provides an identifier for the specific fan and the speed of the fan.
"STC_NodeTemperature" on page 130	This class represents the temperature state of hardware components of a engine. An instance exists for each temperature sensor on every engine in the cluster. It provides current temperature and threshold values.
"STC_NodeVitalProductData" on page 131	This class represents vital product data about the components of a engine. It provides the model and serial number of the host machine and firmware information.
"STC_NodeVoltage" on page 132	This class represents the state of the voltage sources of a engine. It provides engine voltage information and warning thresholds.
"STC_NodeWatchdog" on page 133	This class represents the watchdog for each engine in a cluster.
"STC_TankEvents" on page 154	This class represents a possible event that a server might generate. It provides information about the message that would be logged and the trap that might be generated by the event.
"STC_TankMetrics" on page 155	This class represents the metrics for each subordinate server. It provides metrics for metadata activity, locks, and buffers within the server.
"STC_TankWatchdog" on page 160	This class represents the metadata server restart service operations. It provides the state of the metadata server restart service, probe intervals, total number of retries, total number of absence tests.

Current user and process status classes

The following diagram shows the hierarchy and definitions of the STC_AdminProcess and STC_AdminUser classes.

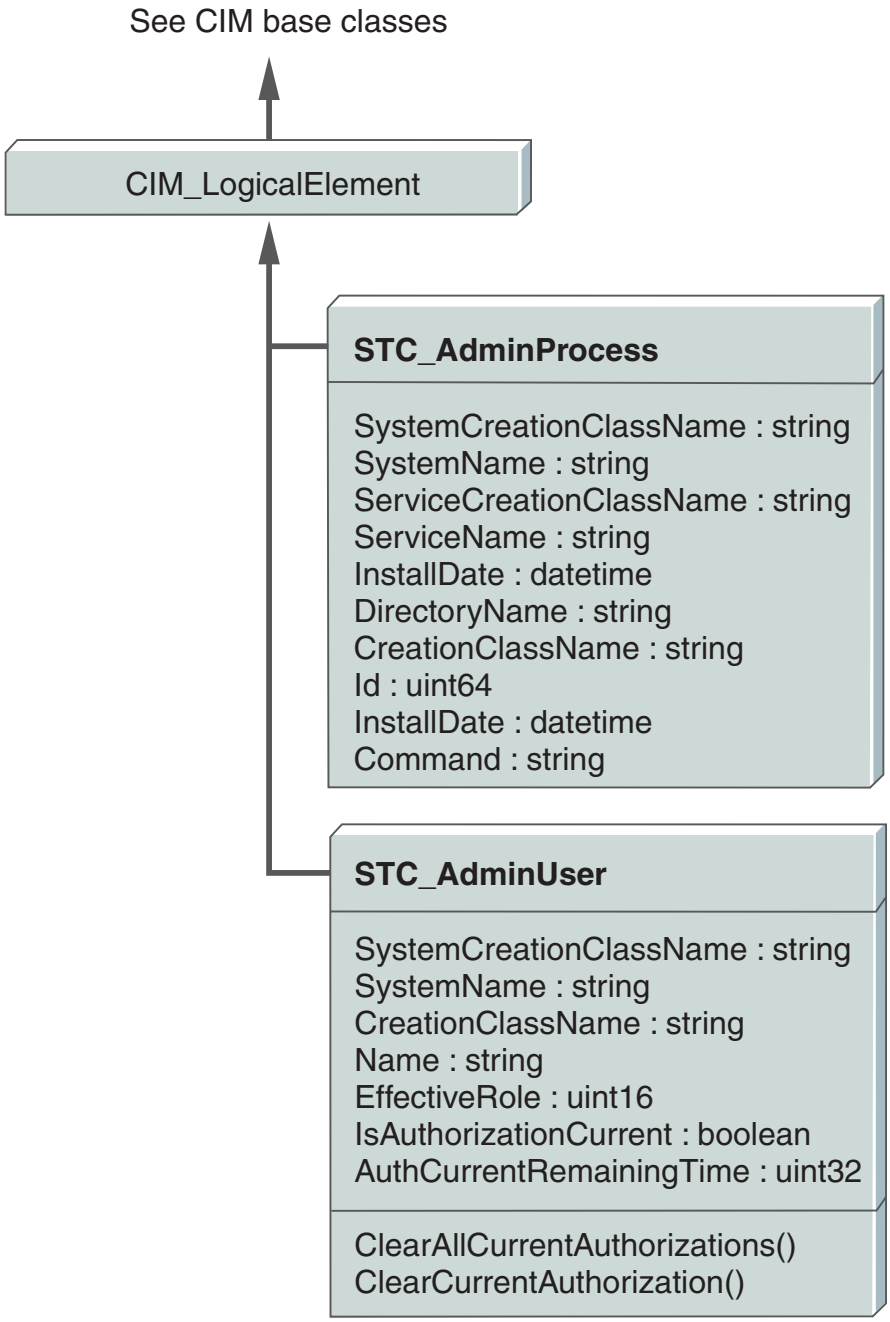


Figure 14. Current user and process status classes

Metrics classes

The following diagram shows the hierarchy and definitions of the STC_MasterMetrics and STC_TankMetrics classes.

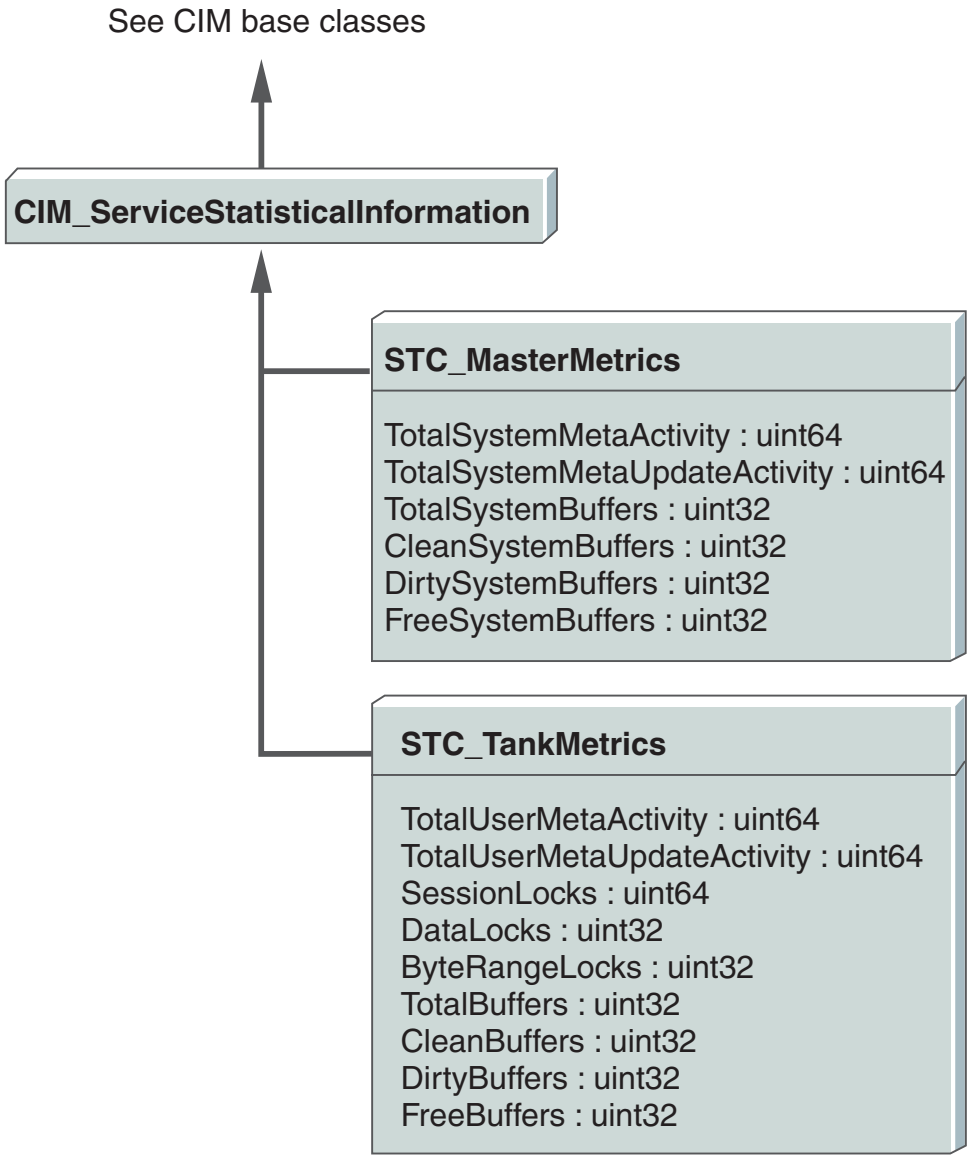


Figure 15. Metrics classes

Engine status classes

The following diagram shows the hierarchy and definitions of the STC_NodeFan, STC_NodeTemperature, STC_NodeVitalProductData, STC_NodeVoltage and STC_NodeWatchdog classes.

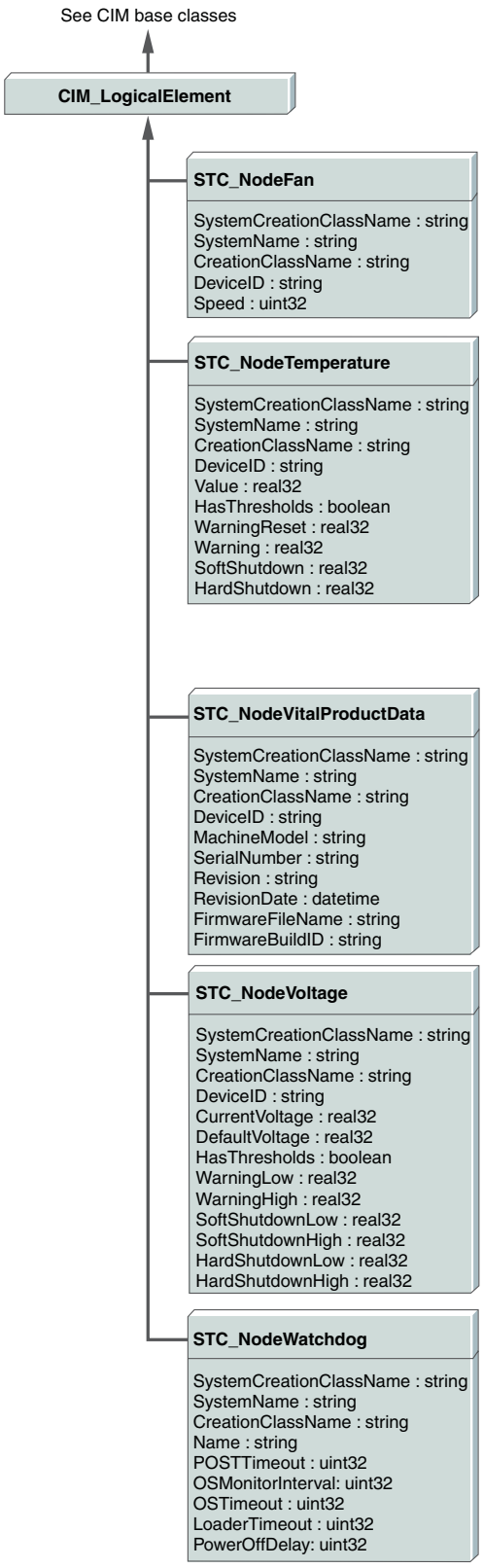


Figure 16. Engine status classes

Server status classes

The following diagram shows the hierarchy and definitions of the STC_TankEvents and STC_TankWatchdog classes.

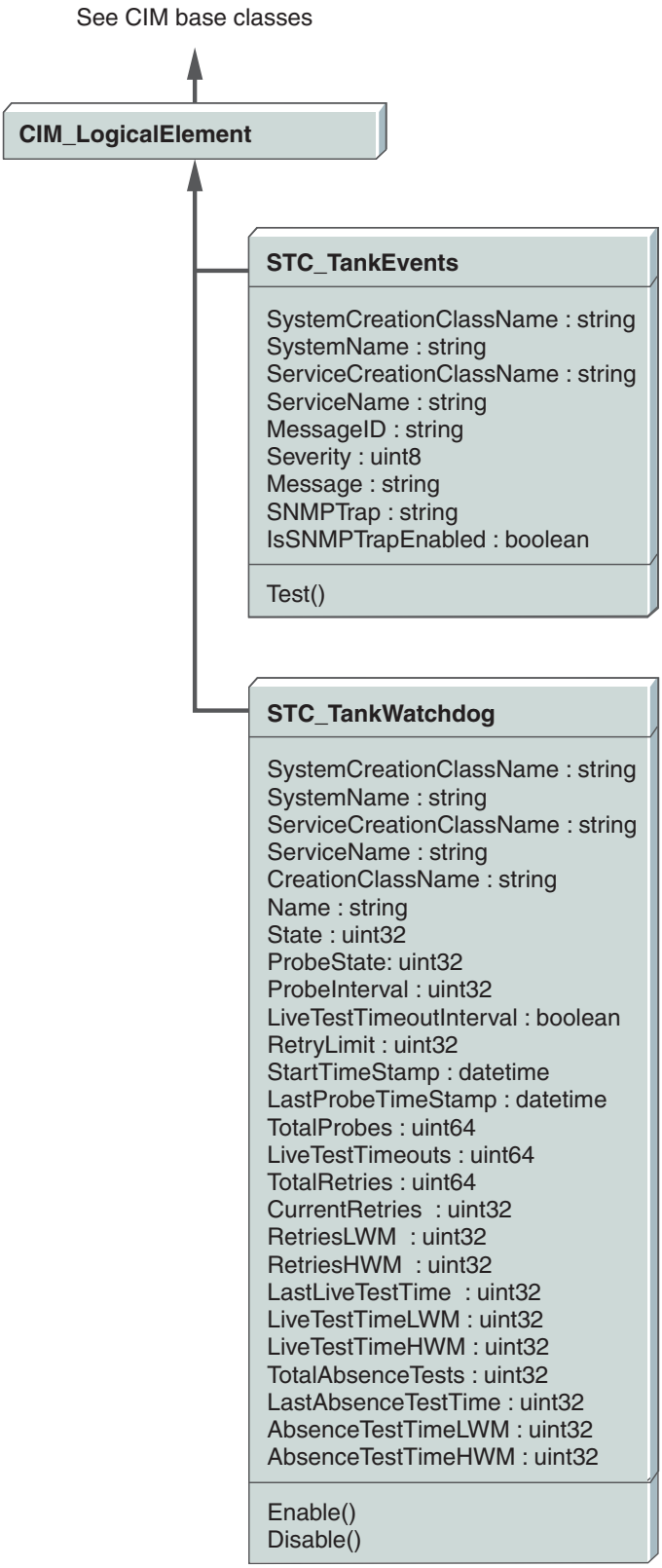


Figure 17. Service status classes

SAN File System log classes

Table 6 provides an overview of the classes that represent logs.

Table 6. SAN File System log classes

Name	Description
"STC_AdminMessageLog" on page 87	This class represents the message log file for the administrative server. It extends the STC_MessageLog class.
"STC_AdminSecurityLog" on page 88	This class represents the security log file for the administrative server. It extends the STC_MessageLog class.
"STC_MDSAuditLog" on page 123	This class represents the audit log file for a metadata server. It extends the STC_MessageLog class.
"STC_MDSEventLog" on page 123	This class represents the event log file for a metadata server. It extends the STC_MessageLog class.
"STC_MDSMessageLog" on page 124	This class represents the message log file for a metadata server. It extends the STC_MessageLog class.
"STC_MessageLog" on page 124	This class represents log files that are present in the SAN File System. It provides identifier and location information about the log. Its methods enable you to traverse a log forwards and backwards and for a specified date and severity levels; retrieve a specified number of log records; and clear a log.

The following diagram shows the hierarchy and definitions of the log classes.

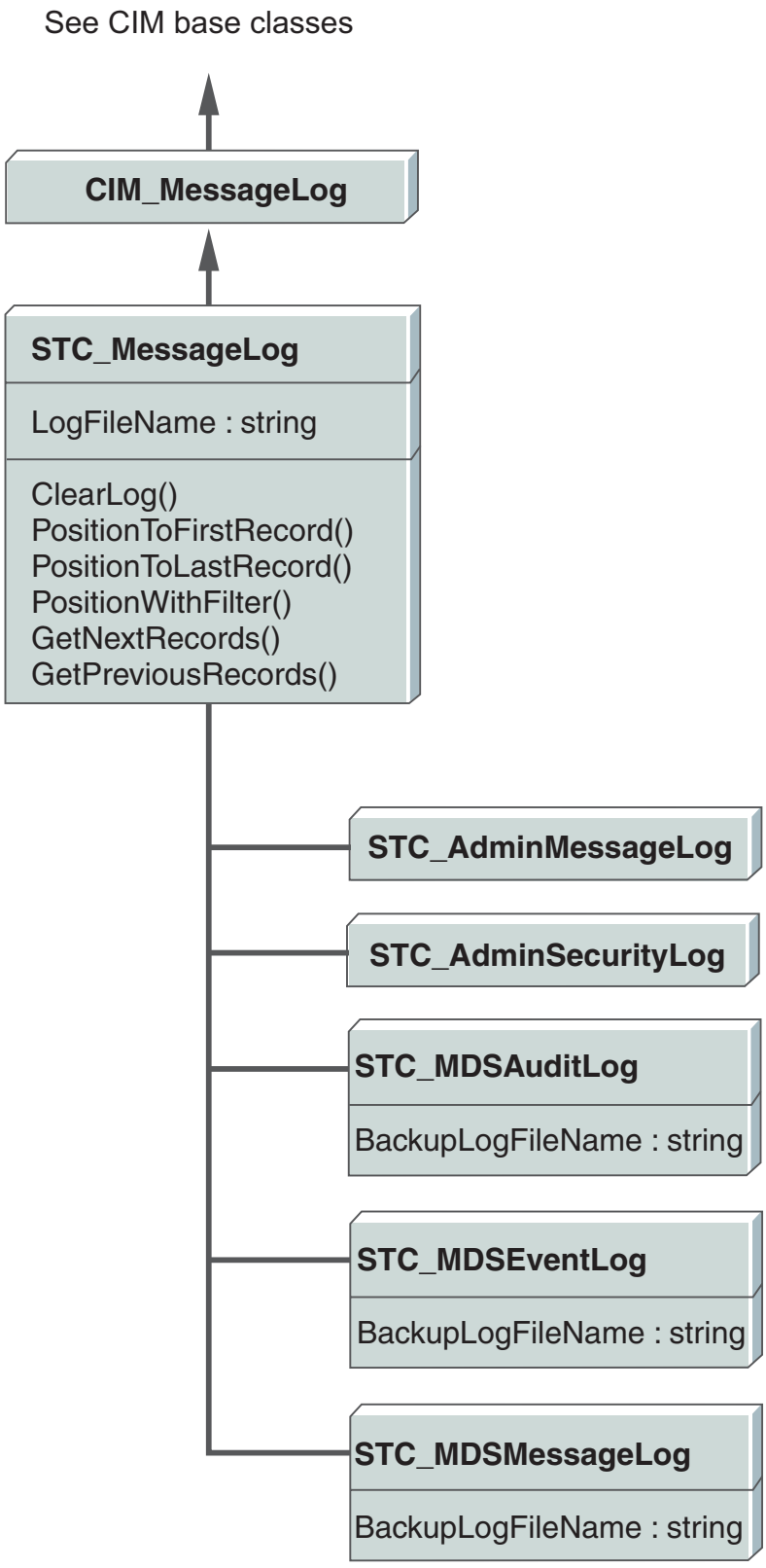


Figure 18. Log classes

SAN File System backup classes

Table 7 provides an overview of the classes for backup.

Table 7. SAN File System backup classes

Name	Description
"STC_PitImage" on page 134	This class represents a FlashCopy image (also known as the point-in-time image) of a fileset. It describes the identifier of the fileset and the FlashCopy image and the FlashCopy image location. Its methods enable you to create a new FlashCopy image, revert the fileset to a FlashCopy image, or delete a FlashCopy image.
"STC_SystemMDRAid" on page 150	This class provides a mechanism to extract system metadata information into a recovery file on a local disk. It provides recovery file identifier, location, and size information, as well as a script to generate commands from the file. Its methods enable you to create and delete a recovery file and to generate commands for recreating metadata from the file.

The following diagram shows the hierarchy and definitions of the backup classes.

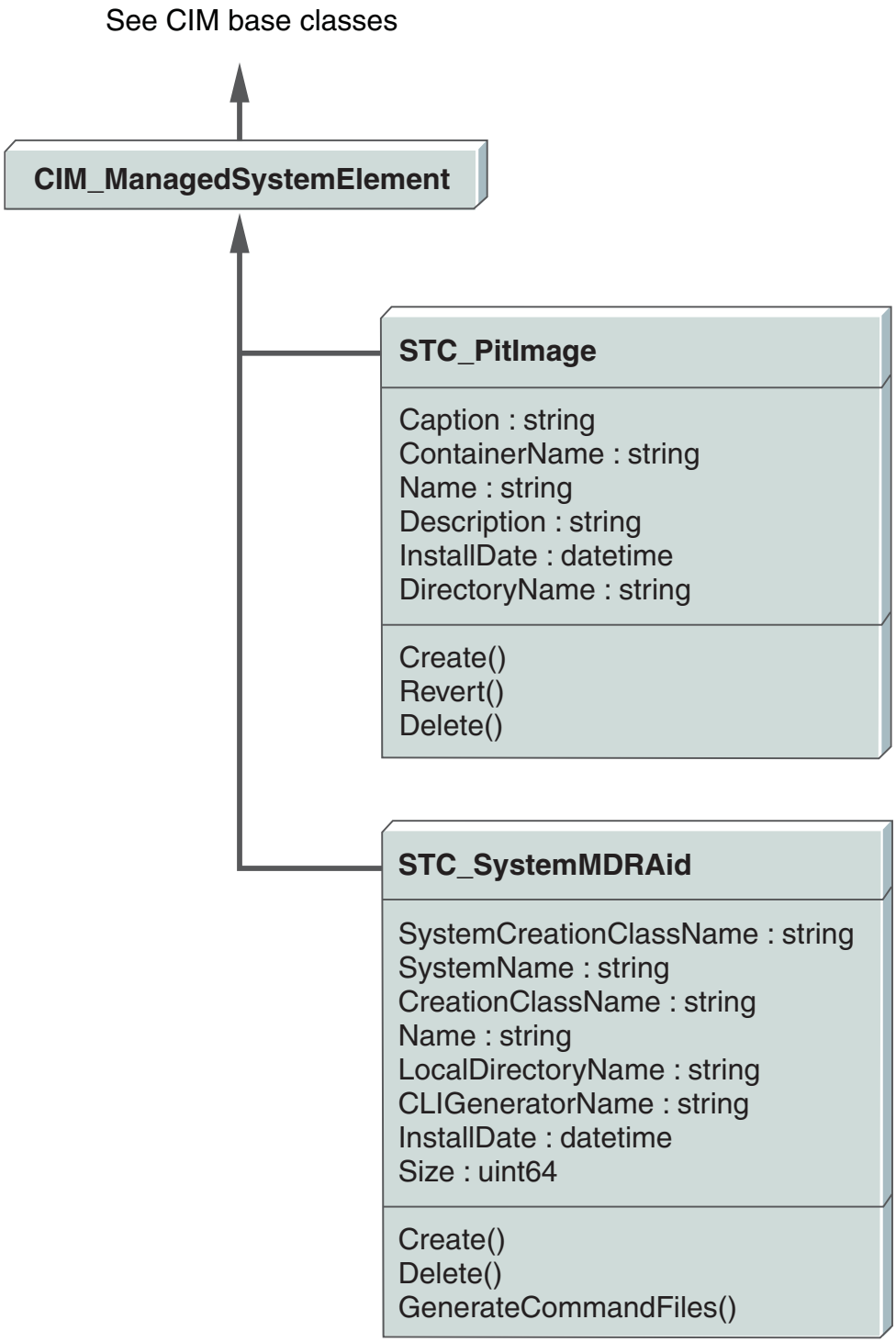


Figure 19. Backup classes

Programming considerations

Third-party CIM clients can manage SAN File System by calling methods in administrative agent classes. In general, CIM clients should interface with the administrative agent on the master metadata server.

Consider the following conventions when programming the classes:

- A CIM client cannot change read-only properties. You can set only those properties that are writable.
- Although method parameters are independent from properties, often a parameter of a method correlates to a property of its class.

Role-based access

For security, the object model implements role-based access. It restricts access to administrative operations through user roles that are stored in the Lightweight Directory Access Protocol (LDAP).

As described in "User roles" on page 31, the Administrator role has access to the Monitor, Backup, and Operator tasks, plus access to Administrator tasks. To define the minimum role needed for a certain operation, the object model uses the following CIM qualifiers:

- ReadRole – The role required to read a property. A CIM client must have at least the Monitor role to read a class property.
- WriteRole – The role required to write a writable property. A CIM client must have the Administrator role to change a writable property.
- ExecuteRole – The role required to invoke a method. A CIM client must have the Administrator role to invoke a method with the following exceptions:
  - CloseClientLUNList() method requires the Monitor role.
  - Create() (FlashCopy image) method requires the Backup role.
  - Create() (recovery file) method requires the Backup role.
  - Delete() (FlashCopy image) method requires the Backup role.
  - Delete() (recovery file) method requires the Backup role.
  - GenerateCommandFiles() method requires the Backup role.
  - GetFileSetInfo() method requires the Backup role.
  - GetNextClientLUN() method requires the Monitor role.
  - GetPolicyRuleStats() method requires the Monitor role.
  - GetStoragePoolStats() method requires the Monitor role.
  - GetRules() method requires the Monitor role.
  - GetNextFOV() method requires the Backup role.
  - GetNextRecords() method requires the Monitor role.
  - GetPreviousRecords() method requires the Monitor role.
  - ListAssociatedPools() method requires the Monitor role.
  - ListClientsByLUN() method requires the Monitor role.
  - ListClientsByVolume() method requires the Monitor role.
  - OneButtonDataCollector() method requires the Operator role.
  - OpenClientLUNList() method requires the Monitor role.
  - PositionToFirstRecord() method requires the Monitor role.
  - PositionToLastRecord() method requires the Monitor role.
  - PositionWithFilter() method requires the Monitor role.

– RescanLUNs() method requires the Monitor role.
– ResetFOV() method requires the Backup role.

Dynamic and static methods

CIM supports dynamic and static methods. A *static method* operates on a class while a *dynamic method* operates on a specific instance of a class. Therefore, a dynamic method must reference a specific instance of the class. You can call a static method by constructing a CIM object path that just has the class name. For a dynamic method, the CIM object path must be the fully-qualified name of the instance.

The fully-qualified name of the instance might include the class name and the following set of keys that uniquely identify the instance:

- CreationClassName - The class name of this instance
- SystemCreationClassName - The class name of the system to which this instance belongs. The system is usually either a cluster (represented by the STC_Cluster class) or an engine (represented by the STC_ComputerSystem class).
- System Name - The instance name of the system to which this instance belongs
- ServiceCreationClassName - The service class is usually either STC_MasterService or STC_TankService.
- Service Name - The instance name of the service

Most extrinsic methods are dynamic.

With a static method, you need to specify only the class name to indicate the CIM object path.

The following extrinsic methods are static methods:

- ClearAllCurrentAuthorizations()
- CloseClientLUNList()
- Create() (fileset)
- Create() (FlashCopy image)
- Create() (policy)
- Create() (recovery file)
- Create() (storage pool)
- Create() (volume)
- CreateUsingLunId()
- DisableDefault()
- GetFileSetInfo()
- GetInstalledLanguages()
- GetNextClientLUN()
- GetPolicyRuleStats()
- GetStoragePoolStats()
- OpenClientLUNList()
- RescanLUNs()
- Test()

Chapter 2. Managing SAN File System

This chapter describes the management of SAN File System using the object model.

Managing clients

This section describes the methods you can invoke to manage clients.

Listing clients by LUN access

You must have Administrator privileges to perform this task.

The STC_MasterService class provides a method for retrieving a list of clients that have access to a specified LUN.

To retrieve the list of client names, invoke the STC_MasterService.ListClientsByLUN() method with the LunID parameter.

Listing clients by volume access

You must have Administrator privileges to perform this task.

The STC_MasterService class provides a method for retrieving a list of clients that have access to a specified volume.

To retrieve the list of client names, invoke the STC_MasterService.ListClientsByVolume() method with the VolumeName parameter.

Managing the cluster

This section describes the methods you can invoke to perform tasks for managing the cluster.

Changing configuration parameters

You must have Administrator privileges to perform this task.

The following SAN File System configuration classes contain writable properties that you can change:

- STC_MasterDisruptiveSetting and STC_MasterDynamicSetting classes contain writable properties that represent cluster configuration parameters.
- STC_LdapDynamicSetting class contains writable properties that represent LDAP configuration parameters.
- STC_RsaDynamicSetting class contains writable properties that represent the Remote Supervisory Adapter (RSA) userid and password, which are used to access information from the engines. The values must match those in the RSA system.
- STC_WatchdogDynamicSetting class contains writable properties that represent metadata-server restart-service tuning options, such as retry limits, probe intervals, and activity test timeouts.

To change these configuration parameters, invoke the SetProperty() intrinsic method. Specify the instance name and the property that represents the configuration parameter.

Changing active cluster states

You must have Administrator privileges to perform this task.

The quiescent states restrict activity on all metadata servers in the cluster.

1. To place the cluster in a quiescent state, invoke the STC_MasterService.QuiesceService() method while specifying the mode parameter. The Mode parameter can be one of the following values:
   a. 0: Partly Quiescent - A limited quiescent mode that allows the client to continue file data activity but prevents client metadata activity and new client connections. In this state, a backup would preserve metadata integrity, but might not preserve file data integrity.
   b. 1: Fully Quiescent - A full quiescent mode that suspends all client metadata activity and file data activity and terminates all client sessions. This state allows a backup with metadata and file data integrity.
   c. 2: Administrative Quiescent - A quiescent mode that allows administrative operations that do not permit client activity.
2. To return the cluster to an active state, invoke the STC_MasterService.ResumeService() method. This method returns the cluster to a fully online state from the quiescent state.

Listing installed languages

You must have Administrator privileges to perform this task.

The STC_MasterDisruptiveSetting class provides a method for retrieving a list of the languages in which a metadata server can run in the cluster. The possible languages are United States English (en_US.utf8) and Japanese (ja_JP.utf8).

To retrieve the list of installed languages, invoke the MasterDisruptiveSetting.GetInstalledLanguages() method.

Starting the cluster

You must have Administrator privileges to perform this task.

The STC_MasterService class provides the method for starting the cluster.

To start the cluster, invoke the STC_MasterService.StartService() method, which brings up all pre-commissioned metadata servers in online mode on all engines. This method starts the master metadata server, verifies that the master metadata server is online, and then starts all subordinate servers. You can also bring up all pre-commissioned metadata servers in offline mode on all engines by invoking the STC_MasterService.StartServiceInAdmin() method.

Stopping the cluster

You must have Administrator privileges to perform this task.

The STC_MasterService class provides the method for stopping a cluster.

To stop a cluster, invoke the STC_MasterService.StopService() method, which brings down all servers on all engines.

Upgrading cluster software

You must have Administrator privileges to perform this task. A cluster upgrade can occur only after you have upgraded each individual metadata server to the same new software version.

The STC_MasterService class provides the method for committing a cluster to start using an upgraded software version.

1. Stop each metadata server in the cluster, install the new version of software, and restart the server.
2. Invoke the STC_MasterService.CommitUpgrade() method to commit the software version upgrade and begin the process of updating the metadata structures.

Managing disaster recovery files

This section describes the methods you can invoke to perform tasks for managing disaster recovery files.

Creating a recovery file

You must have Administrator privileges to perform this task.

The STC_SystemMDRAid class provides the method for creating a cluster-wide, metadata recovery file.

To create a recovery file, invoke the STC_SystemMDRAid.Create() method. If you set the IsForce parameter to True, the new recovery file overwrites any existing one.

Deleting a recovery file

You must have Administrator privileges to perform this task.

The STC_SystemMDRAid class provides the method for deleting a recovery file.

To delete a recovery file, invoke the STC_SystemMDRAid.Delete() method.

Generating recovery commands

You must have Administrator privileges to perform this task.

The STC_SystemMDRAid class provides the GenerateCommandFiles() method for generating recovery commands. The class properties specify the location of the generated command files and the script used to generate command files.

- TankSysCLI.auto - This file contains commands to re-create storage pools, filesets, and policies. In case of disaster, this file can run without manual intervention.
- TankSysCLI.volume - This file contains commands to re-create volumes. This file requires manual verification and editing to run.
- TankSysCLI.attachpoint - This file contains commands to re-create fileset attach points. This file requires manual verification, editing, and intervention to run.

Managing engines

This section describes the methods you can invoke to perform tasks for managing engines.

Powering off the engine

You must have Administrator privileges to perform this task.

The STC_ComputerSystem class provides the SetPowerState() method to power off the engine. Specify the following parameters:

- PowerState - 6 for power off, 7 for hibernate, and 8 for soft off
- Time - The time when the power setting should occur. If value is zero, the setting occurs immediately.

Powering on the engine

You must have Administrator privileges to perform this task.

The STC_ComputerSystem class provides the SetPowerState() method to power on the engine. Specify the following parameters:

- PowerState - 1 for full power, 2 for low power mode, 3 for standby mode, and 4 for any other type of power save
- Time - The time when the power setting should occur. If value is zero, the setting occurs immediately.

Restarting the engine

You must have Administrator privileges to perform this task.

The STC_ComputerSystem class provides the SetPowerState() method to restart the engine. Specify the following parameters:

- PowerState - 5 for power cycle
- Time - The time when the power setting should occur. If value is zero, the setting occurs immediately.

Managing filesets

This section describes the methods you can invoke to perform tasks for managing filesets.

Attaching a fileset

You must have Administrator privileges to perform this task.

The STC_Container class provides the Attach() method to attach a fileset.

- ExistingDirPath - Current path where the fileset directory resides.
- NewDirName - The directory name of the fileset.

Changing the assignment of a fileset server

You must have Administrator privileges to perform this task.

The STC_Container class provides methods to change the metadata server assignment of a fileset.

To change the assignment of a fileset from the metadata server to another specific server, invoke the STC_Container.ChangeServer() method while specifying the new server as a parameter. To remove the static assignment that a fileset has to a server so that the SAN File System dynamically assigns a fileset to a server, invoke the STC_Container.RemoveServerBinding() method.

Creating a fileset

You must have Administrator privileges to perform this task.

The STC_Container class provides the Create() method to create a fileset.

- Name - Your label for the fileset
- Description - Your description of the fileset
- Quota - The maximum size limit, in MB
- IsHardQuota - An indicator that a quota limit cannot be extended
- AlertPercentage - The percent of the fileset size that, when reached, causes the server to generate an alert message
- ExistingDirPath - Current path where the fileset directory resides
- NewDirName - The directory name of the fileset
- Server - The name of the server to host this fileset

Deleting a fileset

You must have Administrator privileges to perform this task.

The STC_Container class provides the method to delete a fileset. You can delete a fileset under the following conditions:

- The fileset is detached
- The fileset is not the global fileset
- The fileset does not have files on it unless the IsForce option is True

To delete a fileset, invoke the STC_Container.Delete() method. If you set the IsForce parameter to True, the Delete() method deletes the fileset even if it has files in it.

Detaching a fileset

You must have Administrator privileges to perform this task.

The STC_Container class provides the method to detach a fileset.

To detach a fileset, invoke the STC_Container.Detach() method. If you set the IsForce parameter to True, the Detach() method detaches the fileset even if clients are using files in it.

Moving a fileset

You must have Administrator privileges to perform this task.

The STC_Container class provides the method to move a fileset.

To move a fileset, invoke the STC_Container.Move() method while specifying your new label for the fileset as a parameter.

Retrieving fileset information

You must have Administrator privileges to perform this task.

The STC_Container class provides the method for retrieving information about the filesets in a cluster.

To retrieve information about filesets, invoke the STC_Container.GetfilesetInfo() method while specifying the names of the filesets about which you want information. This method retrieves the name of the server serving the fileset and transaction statistics.

Managing FlashCopy images

This section describes the methods you can invoke to perform tasks for managing FlashCopy images.

Creating a FlashCopy image

You must have Backup privileges to perform this task.

The STC_PitImage class provides the Create() method to create a FlashCopy image.

- ContainerName - Your label for the fileset to which this FlashCopy image belongs.
- Name - Your administrative name for the FlashCopy image.
- Description - Your description of the fileset.
- DirectoryName - The new directory name to be given to the FlashCopy image.
- IsForce - Indicator of whether to delete the oldest FlashCopy image to create this one when the number of FlashCopy images exceeds the maximum.

Deleting a FlashCopy image

You must have Backup privileges to perform this task.

The STC_PitImage class provides the method to delete a FlashCopy image.

To delete a FlashCopy image, invoke the STC_PitImage.Delete() method. If you set the IsForce parameter to True, the Delete() method deletes the FlashCopy image even if client activity exists.

Reverting to a previous FlashCopy image

Attention: When you revert to a FlashCopy image, all FlashCopy images created after the specified FlashCopy image are deleted. The specified FlashCopy image becomes the primary image for the fileset and no longer appears as an image listed in the .flashcopy directory.

You must have Administrator privileges to perform this task.

The STC_PitImage class provides the method to revert to a previous FlashCopy image.

To revert to a previous a FlashCopy image, invoke the STC_PitImage.Revert() method. If you set the IsForce parameter to True, the Revert() method reverts to the previous image even if client activity exists.

Managing logs

This section describes the methods you can invoke to perform tasks for managing logs.

Clearing logs

You must have Administrator privileges to perform this task.

The STC_MessageLog class provides the method for clearing a message log.

To clear a message log of all entries, invoke the STC_MessageLog.ClearLog() method.

Retrieving log records

You must have Monitor privileges to perform this task.

1. Create an iterator.
   a. To create an iterator and position it at the beginning of the log, invoke the STC_MessageLog.PositionToFirstRecord() method.
   b. To create an iterator and position it at the end of the log, invoke the STC_MessageLog.PositionToLastRecord() method.
   c. To create an iterator and position it at the start or end of messages with a specified date and with specified severities (Information, Warning, Error, Severe), invoke the STCMessageLog.PositionWithFilter() method.
2. Retrieve log records.
   a. To retrieve a specified number of records from the message log starting from the record indicated by the IterationIdentifier parameter, invoke the STC_MessageLog.GetNextRecords() method.
   b. To retrieve a specified number of records from the message log ending from the record indicated by the IterationIdentifier parameter, invoke the STC_MessageLog.GetPreviousRecords() method.

Managing metadata servers

This section describes the methods you can invoke to perform tasks for managing metadata servers.

Changing the master server

You must have Administrator privileges to perform this task.

The STC_TankService class provides the method for making a different server the master metadata server, if the master metadata server is irrecoverably lost. A master can be lost because of hardware failures, software failures, or partitioned networks. Before changing the master metadata server, be aware of the following considerations:

- You cannot change the master when the cluster is down.
- When a master is lost, the subordinate servers are no longer in operational states.

1. Ensure that the previous master is down by turning the power off to the engine hosting the down master metadata server.
2. Invoke the STC_TankService.BecomeMaster() method.
3. Manually log in to the engine and edit the STC.TankService class property to set the IsMaster property to True.

Checking metadata

You must have Administrator privileges to perform this task.

The STC_MasterService class provides a method for checking and repairing metadata. It also provides a method for stopping a metadata check that is in progress.

You can use the FileSystemCheck() method to check and repair metadata. It enables you to specify the following options:

- Check the integrity of the structure and the content of the metadata.
- Check the integrity of the system metadata and the file metadata.
- Limit the user metadata checking to a subset of filesets.

You can restrict this operation to check-only or check and repair. The message log contains a report generated by this method. If you did not limit the mode to check-only, the system automatically salvages and repairs the damaged data if possible. Some types of repair require manual intervention from the administrator. In those cases, the system places the cluster state into Administrative mode.

Note:

1. This method is a long-running process. If there is a cluster reformation while the method is running, this method might stop.
2. Only one FileSystemCheck() operation can be in progress at time.

Specify the following parameters:

- IsCheckOnly - Indicator of whether to only check and not repair.
- CheckScope - A bitmap that indicates the scope of the check, which could be a check of the structure or content or both.

- Type - A bitmap that indicates the type of metadata to be checked. The possible types are system or user (file) or both.
- ContainerList - A list of filesets for the method to check or repair if the Type parameter indicates User and not System.

Retrieving file information

Retrieving file information

You must have Administrator privileges to perform this task.

The STC_MasterService class provides the method for retrieving metadata information for a file.

To retrieve metadata information for a file, invoke the STC_MasterService.GetFileInfo() method, specifying the name of the file about which you want the information. This method retrieves attributes such as the file type and size; the time the file was created, last modified, and last accessed; and the storage pool, fileset, and metadata server on which the file resides.

Starting a metadata server

You must have Administrator privileges to perform this task.

The STC_TankService class provides the method for starting a metadata server on an engine.

To start a metadata server, invoke the STC_TankService.StartService() method.

Starting the metadata server restart service

You must have Administrator privileges to perform this task.

The STC_TankWatchdog class provides the method for enabling the metadata server restart service.

To enable the metadata server restart service, invoke the STC_TankWatchdog.Enable() method.

Stopping a metadata server

You must have Administrator privileges to perform this task.

The STC_TankService class provides the method for stopping a metadata server on an engine.

To stop a metadata server, invoke the STC_TankService.StopService() method.

Stopping the metadata server restart service

You must have Administrator privileges to perform this task.

The STC_TankWatchdog class provides the method for disabling the metadata server restart service.

To disable the metadata server restart service, invoke the STC_TankWatchdog.Disable() method.

Managing policies

This section describes the methods you can invoke to perform tasks for managing policies.

Activating a policy

You must have Administrator privileges to perform this task.

The STC_PolicySet class provides the method for activating a policy.

To activate a policy, invoke the STC_PolicySet.Activate() method.

Creating a policy

You must have Administrator privileges to perform this task.

The STC_PolicySet class provides the Create() method for creating a new policy. Specify the following parameters:

- Name - A label for this policy.
- Description - Your description of this policy.
- PolicyRules - The set of policy rules belonging to this policy.
- IsForce - Indicator of whether to modify an existing policy.

Deleting a policy

You must have Administrator privileges to perform this task.

The STC_PolicySet class provides the method for deleting a policy.

To delete a policy, invoke the STC_PolicySet.Delete() method.

Viewing a policy

You must have Monitor privileges to perform this task.

The STC_PolicySet class provides the method for retrieving the rules associated with a policy.

To retrieve the rules, invoke the STC_PolicySet.GetRules() method. This method returns the rules as a string parameter.

Viewing policy statistics

You must have Administrator privileges to perform this task.

The STC_PolicySet class provides the method for retrieving rule and storage pool statistics about a policy.

To retrieve the policy rule statistics associated with one or more filesets, invoke the STC_PolicySet.GetPolicyRuleStats() method. Specify the Filesets input parameter to indicate the filesets for which you want the statistics. For each fileset, this method returns the rule name, position, storage pool name, and the number of times the rule applied or did not apply to the fileset.

To retrieve the policy storage pool statistics associated with one or more filesets, invoke the STC_PolicySet.GetStoragePoolStats() method. Specify the Filesets input parameter to indicate the filesets for which you want the statistics. For each fileset, this method returns the storage pool name, the files placed in the storage pool, and the time the last file was placed.

Managing storage pools

This section describes the methods you can invoke to perform tasks for managing storage pools.

Creating a storage pool

You must have Administrator privileges to perform this task.

The STC_StoragePool class provides the Create() method for creating new storage pools. Specify the following parameters:

- Name - Your label for the storage pool.
- Description - A string that is your description of the storage pool.
- PartitionSize - The partition size, in megabytes, to use when a fileset allocates space.
- BlockSize - The allocation strategy to use for files on this storage pool.
- AlertPercentage - The percentage of the storage pool size that, when reached, causes the server to generate an alert message.

Deleting a storage pool

You must have Administrator privileges to perform this task.

The STC_StoragePool class provides the method for deleting a storage pool.

To delete a storage pool, invoke the STC_StoragePool.Delete() method.

Disabling the default storage pool

You must have Administrator privileges to perform this task.

The STC_StoragePool class provides the method for disabling the use of a default storage pool.

To disable the use of a default storage pool, invoke the STC_StoragePool.DisableDefault() method.

Moving a storage pool

You must have Administrator privileges to perform this task.

The STC_StoragePool class provides the method to move or rename a storage pool. It creates a new storage pool with the specified name and migrates the data and capabilities to the new name.

To move or rename a storage pool, invoke the STC_StoragePool.Move() method and specify the new label for the storage pool.

Setting the default storage pool

You must have Administrator privileges to perform this task.

The STC_StoragePool class provides the method for changing a user storage pool to the default storage pool. The PoolType property of the STC_StoragePool class changes from User to User Default.

To change the storage pool type from User to User Default, invoke the STC_StoragePool.SetDefault() method.

Managing users

This section describes the methods you can invoke to perform tasks for managing users.

Timing out all user authorizations

You must have Administrator privileges to perform this task.

The STC_AdminUser class provides the method for clearing all current validation windows.

To clear all current validation windows, invoke the STC_AdminUser.ClearAllCurrentAuthorizations() method.

Timing out a user authorization

You must have Administrator privileges to perform this task.

The STC_AdminUser class provides the method for clearing the validation window of a user.

To clear a user's validation window, invoke the STC_AdminUser.ClearCurrentAuthorization() method.

Managing volumes and LUNs

This section describes the methods you can invoke to perform tasks for managing volumes and LUNs.

Activating a suspended volume

You must have Administrator privileges to perform this task.

The STC_Volume class provides the method for resuming suspended partition allocations on a volume.

To activate a suspended volume, invoke the STC_Volume.ResumeAllocation() method.

Adding a volume to a storage pool

You must have Administrator privileges to perform this task.

The STC_Volume class provides Create() and CreateUsingLunId() methods for adding volumes to a storage pool. You add a volume to a storage pool when you create a new volume or move or rename an existing volume.

Listing LUNs

You must have Administrator privileges to perform this task.

1. Establish an iteration of a list of LUNs that a specified client can access, by invoking the STC_Available_LUNs.OpenClientLUNList() method. This method returns an iteration identifier for the list.
2. Invoke the STC_Available_LUNs.GetNextClientLUN() method. This method returns an array of information that includes the LUN ID, vendor, product, version, size, and volume name for each LUN.
3. To refresh the list, invoke the STC_Available_LUNs.RescanLUNs() method, which rescans the LUNs on the host system for a specified client.
4. To close an iteration of a list, invoke the STC_Available_LUNs.CloseClientLUNList() method with the iteration identifier for the list.

Removing volumes from a storage pool

You must have Administrator privileges to perform this task.

The STC_Volume class provides a method for deleting an existing volume.

To delete an existing volume, invoke the STC_Volume.Delete() method. Specify with the IsForce parameter whether the method should delete a volume even if it contains files.

If the IsForce parameter is True, this method deletes all the files that partly or fully exist on the volume before deleting the volume. If the IsForce parameter is False, this method first drains the volume by moving the file data that resides on the volume to other volumes in the same storage pool. If the volume drain fails, the method places the volume into a Suspend Allocations state, which requires manual administrative action.

Resizing a volume

You must have Administrator privileges to perform this task.

The STC_Volume class provides the method for notifying a metadata server that the capacity of the LUN has increased. The notification enables the metadata server to expand the size of the volume.

To enable a metadata server to resize a volume, invoke the STC_Volume.Resize method. When you invoke this method you can specify the name of a client that can access the volume. This parameter is required only if the volume is not

accessible by a metadata server. When this parameter is specified, the method uses the client regardless of whether a metadata server can access the LUN.

Retrieving file entries on a volume

You must have Backup privileges to perform this task.

1. Create an iterator that retrieves each file entry that resides on the volume by invoking the STC_Volume.ResetFOV() method. This method returns the identifier for the iterator as a parameter.
2. Invoke the STC_Volume.GetNextFOV() method. Specify the FOVHandle parameter, which is the file on volume (FOV) iteration identifier.
3. The method returns the following parameters:
   a. FOVHandle - The FOV iteration identifier
   b. FOVEntry - The file entry

Suspending a volume

You must have Administrator privileges to perform this task.

The STC_Volume class provides the method for suspending partition allocations on a volume. A metadata server cannot allocate new data on the volume.

To suspend a volume, invoke the STC_Volume.SuspendAllocation() method.

Collecting problem determination data

You must have Administrator privileges to perform this task.

The STC_ComputerSystem class provides a method to invoke the one-button data collector utility that collects server information and system information that is needed for problem determination.

1. Determine the directory in which the data will collect. By default, the information collects in the /usr/tank/pmf directory. Use the TANKDIR environment variable, if you want to specify a different directory. If the utility generates any stdout output and any stderr output, the output collects in the /tmp/obdcout file on the local disk of the engine.
2. Invoke the STC_ComputerSystem.OneButtonDataCollector() method to invoke the one-button data collector utility.

   See the *Maintenance and Problem Determination Guide* for details about the information the utility collects.

Chapter 3. Administrative agent methods

This chapter describes the intrinsic and extrinsic methods that the administrative agent classes provide. These methods implement the function of the administrative agent.

Intrinsic methods

Intrinsic methods are provided by the Distributed Management Task Force Inc. (DMTF) for the purpose of modeling a typical CIM operation. Intrinsic methods provide the basic means that enable you to work with an object model.

The administrative agent uses the following intrinsic methods:

Table 8. SAN File System Intrinsic Methods

Method name	Functional group
"EnumerateClasses()"	Basic read
"EnumerateClassNames()" on page 76	Basic read
"EnumerateInstanceNames()" on page 77	Basic read
"EnumerateInstances()" on page 77	Basic read
"EnumerateQualifiers()" on page 78	Qualifier declaration
"ExecQuery()" on page 78	Query execution
"GetClass()" on page 78	Basic read
"GetInstance()" on page 79	Basic read
"GetProperty()" on page 79	Basic read
"GetQualifier()" on page 80	Qualifier declaration
"ModifyInstance()" on page 80	Instance manipulation
"SetProperty()" on page 81	Basic write

The CIM intrinsic methods are defined in the *Distributed Management Task Force Inc. (DMTF) Specification for CIM Operations over HTTP* available at www.dmtf.org/standards/documents/WBEM/DSP200.html.

EnumerateClasses()

Use the EnumerateClasses() method to enlist all subclasses of a single object class or all classes of the same object type in the target namespace.

Parameters

You can specify the following parameters of the EnumerateClasses() method:

Table 9. EnumerateClasses() method parameters

Name	Type	Description
ClassName	string	Defines the name of the class for which subclasses are to be returned. If this field is null, all base classes within the target namespace are returned.
DeepInheritance	boolean	If True, returns all subclasses of the specified class. If false, returns only immediate child subclasses.
LocalOnly	boolean	If True, returns all properties, methods, and qualifiers that are overridden within the definition of the class.
IncludeQualifiers	boolean	If True, returns all qualifiers for the class, its properties, methods, or method parameters; if False, returns no qualifiers.
IncludeClassOrigin	boolean	If True, returns the CLASSORIGIN attribute of the class.

The EnumerateClasses() method enumerates the specified one or more classes or returns one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)
- 5 (CIM_ERR_INVALID_CLASS)

EnumerateClassNames()

You can use the EnumerateClassNames() method to enlist the names of all subclasses of a single object class or the names of all classes of the same object type in the target namespace.

Parameters

You can specify the following parameters of the EnumerateClassNames() method:

Table 10. EnumerateClassNames() method parameters

Name	Type	Description
ClassName	string	Defines the name of the class for which subclasses are to be returned. If this field is null, all base classes within the target namespace are returned.
DeepInheritance	boolean	If True, returns all subclasses of the specified class. False returns only immediate child subclasses.

The EnumerateClassNames() method enumerates the specified one or more classes or returns one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)
- 5 (CIM_ERR_INVALID_CLASS)

EnumerateInstanceNames()

You can use the EnumerateInstanceNames() method to enlist all the names of the instances of the same object class in the target namespace.

Parameters

You can specify the following parameters of the EnumerateInstanceNames() method:

Table 11. EnumerateInstanceNames() method parameters

Name	Type	Description
ClassName	string	Defines the name of the class for which instances are to be returned.

The EnumerateInstanceNames() method enumerates the specified names of the instances or returns one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)
- 5 (CIM_ERR_INVALID_CLASS)

EnumerateInstances()

You can use the EnumerateInstances() method to enlist all instances of the same object class in the target namespace.

Parameters

You can specify the following parameters of the EnumerateInstances() method:

Table 12. EnumerateInstances() method parameters

Name	Type	Description
ClassName	string	Defines the name of the class for which instances are to be returned.
DeepInheritance	boolean	If True, returns all instances and all properties of the instance, including those added by creating subclasses. If False, returns only properties defined for the specified class.
LocalOnly	boolean	If True, returns all properties, methods, and qualifiers that are overridden within the definition of the class.
IncludeQualifiers	boolean	If True, returns all qualifiers for each instance, its properties, methods, or method parameters. False returns no qualifiers.
IncludeClassOrigin	boolean	If True, returns the CLASSORIGIN attribute of the class within the instance.

The EnumerateInstances() method enumerates the specified instances or returns one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)

- 5 (CIM_ERR_INVALID_CLASS)

EnumerateQualifiers()

You can use the EnumerateQualifiers() method to enumerate qualifier declarations in the target namespace.

The EnumerateQualifiers() method enumerates the specified qualifier declarations or returns one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)

ExecQuery()

You can use the ExecQuery() method to execute a query against the target namespace.

Parameters

You can specify the following parameters of the ExecQuery() method:

Table 13. ExecQuery() method parameters

Name	Type	Description
QueryLanguage	string	Defines the query language in which the query parameter is expressed. SAN File System supports the WQL Level 1 query language, which is represented by the string WBEMSQL1.
Query	string	Defines the query to be executed.

The ExecQuery() method retrieves one or more classes or instances or returns one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)
- 5 (CIM_ERR_INVALID_CLASS)

GetClass()

You can use the GetClass() method to retrieve a single object class from the target namespace.

Parameters

You can specify the following parameters of the GetClass() method:

Table 14. GetClass() method parameters

Name	Type	Description
ClassName	string	Defines the name of the class to retrieve.
LocalOnly	boolean	If True, returns all properties, methods, and qualifiers overridden within the definition of the class.

Table 14. GetClass() method parameters (continued)

Name	Type	Description
IncludeQualifiers	boolean	If True, returns all qualifiers for the class, its properties, methods, or method parameters. If False, returns no qualifiers.
IncludeClassOrigin	boolean	If True, returns the CLASSORIGIN attribute of the class.

The GetClass() method returns the specified class or one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)

GetInstance()

You can use the GetInstance() method to retrieve a single instance of an object from the target namespace.

Parameters

You can specify the following parameters of the GetInstance() method:

Table 15. GetInstance() method parameters

Name	Type	Description
InstanceName	string	Defines the name of the instance to retrieve.
LocalOnly	boolean	If True, returns all properties, methods, and qualifiers overridden within the definition of the class.
IncludeQualifiers	boolean	If True, returns all qualifiers for the class, its properties, methods, or method parameters. If False, returns no qualifiers.
IncludeClassOrigin	boolean	If True, returns the CLASSORIGIN attribute of the class.

The GetInstance() method returns the specified class or one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)
- 5 (CIM_ERR_INVALID_CLASS)
- 6 (CIM_ERR_NOT_FOUND)

GetProperty()

You can use the GetProperty() method to retrieve a single attribute value of an instance in the target namespace.

Parameters

You can specify the following parameters of the GetProperty() method:

Table 16. GetProperty() method parameters

Name	Type	Description
InstanceName	string	Defines the name of the instance.
PropertyName	string	Defines the name of the property whose value is to be returned from the instance.

The GetProperty() method returns the specified property of the target instance or one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)
- 5 (CIM_ERR_INVALID_CLASS)
- 6 (CIM_ERR_NOT_FOUND)
- 12 (CIM_ERR_NO_SUCH_PROPERTY)

GetQualifier()

You can use the GetQualifier() method to retrieve a single qualifier declaration from the target namespace.

Parameters

You can specify the following parameters of the GetQualifier() method:

Table 17. GetQualifier() method parameters

Name	Type	Description
QualifierName	string	Defines the qualifier whose declaration is to be returned.

The GetQualifier() method returns the specified qualifier or returns one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)
- 6 (CIM_ERR_NOT_FOUND)

ModifyInstance()

You can use the ModifyInstance() method to modify an existing instance of an object in the target namespace.

Parameters

You can specify the following parameters of the ModifyInstance() method:

Table 18. ModifyInstance() method parameters

Name	Type	Description
ModifiedInstance	string	Defines the name of the instance to modify
IncludeQualifiers	boolean	True modifies qualifiers.
propertyList	string[]	Specifies specific properties that are to be modified. If NULL is specified, all properties are modified.

The ModifyInstance() method returns the specified class or one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)
- 5 (CIM_ERR_INVALID_CLASS)
- 6 (CIM_ERR_NOT_FOUND)

SetProperty()

You can use the SetProperty() method to define a single property value of an instance in the target namespace. You can use it to change a configuration parameter value by changing the writable property in an instance of the STC_MasterDisruptiveSetting or STC_MasterDynamicSetting class.

Parameters

You can specify the following parameters of the SetProperty() method:

Table 19. SetProperty() method parameters

Name	Type	Description
InstanceName	string	Defines the name of the instance.
Property	string	Defines the name of the property whose value is to be defined.
NewValue		Defines the new value for the Property (which might be NULL).

The SetProperty() method defines the property name of the target instance or returns one of the following error codes:

- 1 (CIM_ERR_FAILED)
- 2 (CIM_ERR_ACCESS_DENIED)
- 3 (CIM_ERR_INVALID_NAMESPACE)
- 4 (CIM_ERR_INVALID_PARAMETER)
- 5 (CIM_ERR_INVALID_CLASS)
- 6 (CIM_ERR_NOT_FOUND)
- 12 (CIM_ERR_NO_SUCH_PROPERTY)
- 13 (CIM_ERR_TYPE_MISMATCH)

Intrinsic method return codes

For easier diagnosis, return codes from method invocations have the same general meanings.

Table 20 describes the meanings of intrinsic method return codes.

Table 20. Intrinsic method return codes

Code	Symbolic Name	Definition
1	CIM_ERR_FAILED	A general error occurred that is not covered by a more specific error code.
2	CIM_ERR_ACCESS_DENIED	Access to a CIM resource was not available to the client.
3	CIM_ERR_INVALID_NAMESPACE	The target namespace does not exist.
4	CIM_ERR_INVALID_PARAMETER	One or more parameter values passed to the method were invalid.
5	CIM_ERR_INVALID_CLASS	The specified class does not exist.
6	CIM_ERR_NOT_FOUND	The requested object could not be found.
12	CIM_ERR_NO_SUCH_PROPERTY	The specified property does not exist.
13	CIM_ERR_TYPE_MISMATCH	The value supplied is incompatible with the type.

When you invoke an intrinsic method, the administrative agent returns more specific information along with the intrinsic method return code. The administrative agent includes a string in the form SSG:nn where nn is an extrinsic method return code. For example, if you invoke EnumerateInstances() method on the STC_Container class when the server is down, you receive the CIM_ERR_FAILED return code as well as the string SSG:65. The 65 represents the return code meaning: "Server not available". See "Extrinsic method return codes" on page 84 for a description of these return code meanings.

Extrinsic methods

Extrinsic methods are specific to administrative- agent object classes. They add function to the object classes.

The administrative agent supports the following extrinsic methods. Each extrinsic method is described with its object class in the Chapter 4, "Administrative agent object classes," on page 87 chapter in this document.

Table 21. SAN File System extrinsic methods

Method origin (derived from)	Method name
STC_AdminUser	"ClearAllCurrentAuthorizations() method" on page 89
	"ClearCurrentAuthorization() method" on page 89
STC_AvailableLUNs	"CloseClientLUNList() method" on page 90
	"GetNextClientLUN() method" on page 91
	"OpenClientLUNList() method" on page 92
	"RescanLUNs() method" on page 93

Table 21. SAN File System extrinsic methods (continued)

Method origin (derived from)	Method name
STC_ComputerSystem	"GetPowerState() method" on page 95
	"OneButtonDataCollector() method" on page 96
	"SetPowerState() method" on page 96
STC_Container	"Attach() method" on page 99
	"ChangeServer() method" on page 100
	"Create() method" on page 101
	"Delete() method" on page 102
	"Detach() method" on page 103
	"GetFileSetInfo() method" on page 104
	"ListAssociatedPools() method" on page 105
	"Move() method" on page 105
	"RemoveServerBinding() method" on page 106
STC_MasterDisruptiveSetting	"GetInstalledLanguages() method" on page 109
STC_MasterService	"AddServer() method" on page 112
	"CommitUpgrade() method" on page 113
	"DropServer() method" on page 114
	"DropServerByName() method" on page 115
	"FileSystemCheck() method" on page 115
	"GetFileInfo() method" on page 117
	"ListClientsByLUN() method" on page 118
	"ListClientsByVolume() method" on page 119
	"QuiesceService() method" on page 119
	"ResumeService() method" on page 120
	"StartService() method" on page 121
	"StartServiceInAdmin() method" on page 121
	"StopFileSystemCheck() method" on page 122
	"StopService() method" on page 122
STC_MessageLog	"ClearLog() method" on page 124
	"GetNextRecords() method" on page 125
	"GetPreviousRecords() method" on page 126
	"PositionToFirstRecord() method" on page 127
	"PositionToLastRecord method" on page 128
	"PositionWithFilter() method" on page 129
STC_PitImage	"Create() method" on page 135
	"Delete() method" on page 136
	"Revert() method" on page 137

Table 21. SAN File System extrinsic methods (continued)

Method origin (derived from)	Method name
STC_PolicySet	"Activate() method" on page 139
	"Create() method" on page 139
	"Delete() method" on page 140
	"GetPolicyRuleStats() method" on page 141
	"GetRules() method" on page 142
	"GetStoragePoolStats() method" on page 142
STC_StoragePool	"Create() method" on page 147
	"Delete() method" on page 148
	"DisableDefault() method" on page 148
	"Move() method" on page 149
	"SetDefault() method" on page 149
STC_SystemMDRAid	"Create() method" on page 151
	"Delete() method" on page 151
	"GenerateCommandFiles() method" on page 152
STC_TankEvents	"Test() method" on page 155
STC_TankService	"BecomeMaster() method" on page 158
	"StartService() method" on page 159
	"StopService() method" on page 160
STC_TankWatchdog	"Enable() method" on page 163
	"Disable() method" on page 163
STC_Volume	"Create() method" on page 164
	"CreateUsingLunId() method" on page 166
	"Delete() method" on page 167
	"DeleteUsingClient() method" on page 168
	"GetNextFOV() method" on page 169
	"Move() method" on page 170
	"ResetFOV() method" on page 171
	"Resize() method" on page 172
	"ResumeAllocation() method" on page 172
	"SuspendAllocation() method" on page 173

Extrinsic method return codes

For easier diagnosis, return codes from method invocations have the same general meanings. Return codes from extrinsic methods have the following meanings:

- 0 - Method completed successfully
- 1 - Not supported
- 2 - Access failed
- 3 - Already defined or already disabled
- 4 - Command failed
- 5 - In use

- 7 - Insufficient space
- 8 - Integrity lost
- 9 - Name not valid
- 10 - Invalid parameter
- 11 - Invalid size
- 12 - I/O failed
- 13 - Is Default
- 14 - Is referenced
- 15 - Is System
- 18 - Already exists
- 20 - Not attached
- 21 - Not found
- 22 - Not the primary administrative server
- 23 - Not viable
- 24 - Server timed out
- 25 - Policy bind errors
- 26 - Policy syntax error
- 27 - Is global fileset
- 28 - Storage pool not found
- 30 - Transaction failed
- 32 - Volume in use
- 33 - Volume not found
- 34 - Allocations already suspended
- 35 - Allocations were not suspended
- 36 - Is attached
- 37 - End of iteration
- 38 - Invalid iteration identifier
- 39 - File not found
- 40 - Cannot read file
- 41 - Partial data
- 43 - Directory exists
- 44 - Incompatible operation
- 45 - Server not found
- 46 - Invalid cluster state
- 52 - Disk not viable
- 56 - Access denied
- 57 - No space
- 61 - Cannot connect to server
- 62 - Too many connections
- 63 - Metadata server restart service is already enabled
- 64 - Metadata server restart service is already disabled
- 65 - Server not available
- 66 - Metadata server restart service state cannot continue
- 67 - Cannot become the primary administrative server
- 68 - Already in progress

- 69 - Up-to-date
- 70 - Servers not the same version
- 71 - RSA unavailable
- 75 - Invalid value
- 76 - Cancel pending
- 77 - Salvage failed
- 78 - Aborted - One or more metadata servers, including the master, has aborted.
- 79 - System volume
- 80 - Device not found
- 81 - Client not found
- 82 - Client not reachable
- 83 - Client I/O failed
- 84 - Client unsupported operation
- 85 - File system check resized
- 86 - Server unsupported operation
- 87 - Not a directory
- 88 - Rogue client
- 89 - Invalid LUN ID
- 91 - Cannot contact remote CIM agent

Chapter 4. Administrative agent object classes

This chapter describes the classes that make up the administrative agent object model. The object classes are the building blocks of the administrative agent and provide management function to the SAN File System.

STC_AdminMessageLog

The STC_AdminMessageLog class represents the aggregated, message log file for the administrative server. This class extends the STC_MessageLog class. It inherits methods from the STC_MessageLog class that enable you to traverse the log and retrieve a specified number of log records.

STC_AdminProcess

The STC_AdminProcess class represents the long-running administrative commands in the cluster. This class extends the CIM_LogicalElement class.

Properties

The STC_AdminProcess class has the following properties:

Table 22. STC_AdminProcess class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_Cluster. This property is *key*. A key property is a property that is used to provide a unique identifier for an instance of a class. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
ServiceCreationClassName	string	The class name of the scoping service: STC_MasterService. This property is key. The maximum length is 256 characters.
ServiceName	string	The instance name of the scoping service. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_AdminProcess. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
Id	uint64	The identifier of the process. This property is key.
InstallDate	datetime	The date-and-time timestamp for the beginning of the process. This property is read-only.
Command	string	The command that initiated this process. The Command string contains a command name and a list of parameters. This property is read-only. The maximum length is 256 characters.

STC_AdminSecurityLog

The STC_AdminSecurityLog class represents the aggregated, security log file for the administrative server. This class extends the STC_MessageLog class. It inherits methods from the STC_MessageLog class that enable you to traverse the log and retrieve a specified number of log records.

STC_AdminUser

The STC_AdminUser class represents an authorized user of SAN File System. This class extends the CIM_LogicalElement class.

The Common Information Model Object Model (CIMOM) authenticates a user by comparing a user name and password with information stored in the LDAP server. After the CIMOM authenticates a user, it authorizes the user depending on whether the user has the appropriate level of access to perform a requested action. Every property and method of a class has a minimum role needed to get or set a property or invoke a method.

When CIMOM receives an administrator request, it authenticates the user in the LDAP server and extracts the role of a successfully authenticated user. For performance purposes, an authenticated user's role remains validated for a small interval of time. CIMOM does not consult LDAP again within this time window. If the same user makes an administrator request again when the window is open, the CIMOM authenticates the user without consulting the LDAP server.

Properties

The STC_AdminUser class has the following properties:

Table 23. STC_AdminUser class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_Cluster. This property is key. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_AdminUser. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
Name	string	The user name. This property is key. The maximum length is 256 characters.
EffectiveRole	uint16	The effective role of the user, as determined by the authentication (CIMOM) service. The direct role of the user or the groups to which the user belongs determine the user's effective role. The strongest role is the effective role. Possible values are: 0: Administrator 1: Operator 2: Backup 3: Monitor Any other value indicates that the role is unknown.

Table 23. STC_AdminUser class properties (continued)

Name	Type	Description
IsAuthorizationCurrent	boolean	An indicator of whether the role is currently validated. If a user's role was recently validated on the LDAP server, the effective role for the user will remain valid for a short interval of time. If a user makes a request again in this time interval, the LDAP server will not be contacted for authentication.
AuthCurrentRemainingTime	uint32	The remaining time interval after which if the user makes a request, the LDAP server will be contacted again to determine the effective role of a user. The value of this property is valid only if validation is current; otherwise, the value will be zero.

ClearAllCurrentAuthorizations() method

Use the ClearAllCurrentAuthorizations() method to clear all current validation windows.

Execute Role: Administrator

Method Type: Static

Return values

The ClearAllCurrentAuthorizations() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 21 (Not found)
- 30 (Transaction failed)
- .. (Internal error)

ClearCurrentAuthorization() method

Use the ClearCurrentAuthorization() method to clear a user's validation window.

Execute Role: Administrator

Method Type: Dynamic

Return values

The ClearCurrentAuthorization() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 21 (Not found)
- 30 (Transaction failed)
- .. (Internal error)

STC_AvailableLUNs

The STC_AvailableLUNs class represents available Fibre Channel logical unit numbers (LUNs). These LUNs are the storage volumes exposed using the small computer system interface (SCSI) LUNs on the engines of the cluster. The STC_AvailableLUNs class extends the CIM_StorageVolume.

Properties

The STC_AvailableLUNs class has the following properties:

Table 24. STC_AvailableLUNs class properties

Name	Type	Description
LunID	uint64	The LUN identifier.
NodeWWN	string	The engine worldwide name (WWN) providing the LUN. A 16-digit hexidecimal number in the form xx:xx:xx:xx:xx:xx:xx:xx. The maximum length is 23 characters. **Note:** This property is not available in SAN File System version 2.1 and does not contain valid data.
PortWWN	string	The port WWN on the engine that is providing the LUN. A 16-digit hexidecimal number in the form xx:xx:xx:xx:xx:xx:xx:xx. The maximum length is 23 characters. **Note:** This property is not available in SAN File System version 2.1 and does not contain valid data.
Vendor	string	The name of the vendor supplying the product. The maximum length is 256 characters.
Product	string	The product name. The maximum length is 256 characters.
Version	string	The version of the product. The maximum length is 256 characters.
Size	uint64	Storage size, in megabytes, of the LUN.
State	uint32	The availability state of this LUN as a volume. Possible values are: 0: Available - This LUN is available to be added as a volume. 1: Assigned - This LUN is already assigned to SAN File System as a volume. The VolumeName property value identifies the specific volume. 2: Error - An error occurred determining the properties of the LUN. 3: Unknown - The Metadata server is not running. Cannot determine the availability of the LUN. 4: Unusable - This LUN is unsuitable as a volume. One reason the LUN is not suitable is that the (inherited) Access property shows the LUN does not support read/write. Other reasons include inconsistent availability of this LUN from all engines of the cluster.
VolumeName	string	If the LUN is already assigned to a SAN File System storage pool, this is the volume name that was given when the LUN was added. Otherwise, the value will be null. The maximum length is 256 characters.

CloseClientLUNList() method

Use the CloseClientLUNList() method to close an iteration of a list of LUNs accessed by a specific client.

Execute Role: Monitor

Method Type: Static

Parameters

Table 25 describes the parameters that you can specify for the CloseClientLUNList() method.

Table 25. CloseClientLUNList() method parameters

Name	Type	Description
IterationIdentifier	string	The input/output parameter that is an identifier for the iterator.

Return values

The CloseClientLUNList() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 30 (Transaction failed)
- 38 (Invalid IterationIdentifier)
- .. (Internal error)

GetNextClientLUN() method

Use the GetNextClientLUN() method to retrieve information for the next LUN in a list of LUNs accessed by a specific client.

This method returns information about one or more LUNs. It uses a set of array output parameters to represent information about each LUN.

Execute Role: Monitor

Method Type: Static

Parameters

Table 26 describes the parameters that you can specify for the GetNextClientLUN() method.

Table 26. GetNextClientLUN() method parameters

Name	Type	Description
IterationIdentifier	string	Input/output parameter that is an identifier for the iterator for the current iteration.
NumberOfEntries	uint32	Input/output parameter that indicates the number of entries to be retrieved and returns the actual number of entries that were retrieved.
LunID	string[]	Output parameter with the identifier of the LUN.
Vendor	string[]	Output parameter that is the name of the vendor supplying the product. The maximum length is 256 characters.
Product	string[]	Output parameter that is the product name. The maximum length is 256 characters.
Version	string[]	Output parameter that is the version of the product. The maximum length is 256 characters.
BlockSize	uint64[]	Output parameter that is the size in bytes of blocks.

Table 26. GetNextClientLUN() method parameters (continued)

Name	Type	Description
NumberOfBlocks	uint64[]	Output parameter that is the total number of logically contiguous blocks.
Size	uint64[]	Output parameter that contains the storage size, in MB, of the LUN.
DeviceID	string[]	Output parameter that is identifying information to uniquely name the LUN.
State	uint32[]	Output parameter that is the availability state of this LUN as a volume. Possible values are: 0: Available - This LUN is available to be added as a volume. 1: Assigned - This LUN is already assigned to SAN File System as a volume. The VolumeName property value identifies the specific volume. 2: Error - An error occurred determining the properties of the LUN. 3: Unknown - The metadata server is not running. The availability of the LUN cannot be determined. 4: Unusable - This LUN is unsuitable as a volume. The LUN may be unsuitable because the (inherited) Access property shows that the LUN does not support read/write or because it is inconsistently available from all engines of the cluster.
VolumeName	string[]	Output parameter that is the volume name that was given when the LUN was added, if the LUN is already assigned to a storage pool. Otherwise, the value will be null. The maximum length is 256 characters.

Return values

The GetNextClientLUN() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 10 (Invalid parameter)
- 30 (Transaction failed)
- 37 (End of iteration)
- 38 (Invalid IterationIdentifier)
- .. (Internal error)

OpenClientLUNList() method

Use the OpenClientLUNList() method to establish an iteration of a list of LUNs accessed by a specific client.

Execute Role: Monitor

Method Type: Static

Parameters

Table 27 on page 93 describes the parameters that you can specify for the OpenClientLUNList() method.

Table 27. OpenClientLUNList() method parameters

Name	Type	Description
IterationIdentifier	string	The output parameter that is an identifier for the iterator.
ClientName	string	The input parameter that is the client name for which the list of LUNs is to be retrieved.

Return values

The OpenClientLUNList() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 10 (Invalid parameter)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 81 (Client not found)
- 82 (Client not reachable)
- 84 (Client unsupported operation)
- .. (Internal error)

RescanLUNs() method

Use the RescanLUNs() method to rescan the LUNs on the host system.

Execute Role: Monitor

Method Type: Static

Parameters

Table 28 describes the parameters that you can specify for the RescanLUNs() method.

Table 28. RescanLUNs() method parameters

Name	Type	Description
ClientName	string	Input parameter that is the client name for which the ReScan operation needs to take place. If the client name is null, this operation will take place on the local metadata server.

Return values

The RescanLUNs() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)

- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 81 (Client not found)
- 82 (Client not reachable)
- 84 (Client unsupported operation)
- .. (Internal error)

STC_Cluster

The STC_Cluster class, along with the STC_MasterService class, provides cluster operations. It extends the CIM_Cluster class.

Properties

The STC_Cluster class has the following properties:

Table 29. STC_Cluster class properties

Name	Type	Description
CreationClass Name	string	The name of the class or the subclass used in the creation of an instance: STC_Cluster. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
ClusterId	uint32	A unique integer identifier for the cluster.
ConfiguredNumberOfNodes	uint32	The number of configured engines in the cluster.
CurrentNumberOfNodes	uint32	Number of engines participating in the cluster. If this number is not the same as the configured number, the cluster is not operating at its full potential.

STC_ComputerSystem

The STC_ComputerSystem class represents each engine in the cluster. This class extends the CIM_ComputerSystem class.

Properties

The STC_ComputerSystem class has the following properties.

Table 30. STC_ComputerSystem class properties

Name	Type	Description
CreationClass Name	string	The name of the class or the subclass used in the creation of an instance: STC_ComputerSystem. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
TotalPowerOnHours	uint64	The total number of hours the engine has been powered on. This is a read-only counter property.
RestartCount	uint16	The total number of times the engine has been power cycled. This is a read-only counter property.

Table 30. STC_ComputerSystem class properties (continued)

Name	Type	Description
IsPowerOn	boolean	An indicator of whether the engine is powered on. This property is read-only.
ASMTime	datetime	The current time on the Advanced System Management Processor's local clock. It is the time reference that must be used to schedule a power off using the "SetPowerState() method" on page 96. This time is independent of the date and time on the server. This property is read-only.
CurrentState	uint32	The state of the system. This property is read-only. Possible values are: • 0: Unknown/Power Off • 1: In Power-On Self Test (POST) • 2: Stopped in POST • 3: Booted Flash • 4: Booting operating system • 5: In operating system • 6: CPU is held in reset • 7: Before POST
UUID	string	The universal unique identifier of the engine. This string represents a 128-bit number in the form of 32 consecutive hexadecimal numbers with no delimiters. This property is read-only.

GetPowerState() method

Use the GetPowerState() method to retrieve the power state of an engine

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 31 describes the parameters that you can specify for the GetPowerState() method.

Table 31. GetPowerState() method parameters

Name	Type	Description
CurrentState	uint32	Output parameter that is the current power setting of the engine. Possible values are: • 0: Unknown/Power Off • 1: In Power-On Self Test (POST) • 2: Stopped in POST • 3: Booted Flash • 4: Booting operating system • 5: In operating system • 6: CPU is held in reset • 7: Before POST

Return values

The GetPowerState() method returns one of the following codes:

- 0 (Completed successfully)
- 4 (Command failed)
- 8 (Integrity lost)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 71 (RSA unavailable)
- .. (Internal error)

OneButtonDataCollector() method

Use the OneButtonDataCollector() method to invoke the one-button data collector utility that collects server and system information needed for problem determination. See the *Maintenance and Problem Determination Guide* for details about the information that the utility collects.

By default, the information collects in the /usr/tank/pmf directory. You can specify a different directory using the TANKDIR environment variable. Any stdout and stderr output generated by the utility collects in the /tmp/obdcout file on the local disk of the engine.

Execute Role: Administrator

Method Type: Dynamic

Return values

The OneButtonDataCollector() method returns one of the following codes:

- 0 (Completed successfully)
- .. (Internal error)

SetPowerState() method

Use the SetPowerState() method to set the power state of the engine. This class overrides the SetPowerState() method in its parent class, CIM_ComputerSystem. It supports only a limited subset of power setting capabilities that are fully described in the parent class.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 32 on page 97 describes the parameters that you can specify for the SetPowerState() method.

Table 32. SetPowerState() method parameters

Name	Type	Description
PowerState	uint16	The input parameter that is the power setting of the engine. Possible values are: • 1: Full Power • 2: Power Save - Low Power Mode • 3: Power Save - Standby • 4: Power Save - Other • 5: Power Cycle • 6: Power Off • 7: Hibernate • 8: Soft Off
Time	datetime	The input parameter that is the time that the power setting should occur. If the value is zero, the setting occurs immediately.

Return values

The SetPowerState() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 4 (Command failed)
- 8 (Integrity lost)
- 30 (Transaction failed)
- 71 (RSA unavailable)
- .. (Internal error)

STC_Container

The STC_Container class represents a fileset (also known as a container). It extends the CIM_ManagedSystemElement class.

There is an instance of this class for every fileset that exists in a SAN File System. The list of instances and the methods defined in this class are available only on the master Metadata server.

Properties

The STC_Container class has the following properties:

Table 33. STC_Container class properties

Name	Type	Description
Name	string	Your label for the fileset when you create, move, or rename it. This property is key. The maximum length is 256 characters.
Description	string	Your description of the fileset. This property is writable. The maximum length is 256 characters.
InstallDate	datetime	The time when the fileset was created. A lack of a value does not indicate that the fileset does not exist. This property is read-only.

Table 33. STC_Container class properties (continued)

Name	Type	Description
State	uint32	The state of the fileset. This property is read-only. Possible values are: 0: Detached 1: Attached
AttachPoint	string	The attach point of this fileset in the file system namespace. An attach point is the combined path formed by the directory path used to attach and the directory name. This property gives the fully qualified directory name of this fileset. It combines the DirectoryPath property with the DirectoryName property. If the fileset is not attached, this value is null. This property is read-only.
DirectoryName	string	The name of the fileset as known to the file system. A fileset is made available to the file system using a different name than the Name property, called the directory name. A directory name is attached to an existing directory path that can be another fileset's attach point or a file directory. This name will appear as a directory under the path shown by DirectoryPath property. You specify this name when you create the fileset and can change it by reattaching the fileset. This property is read-only.
DirectoryPath	string	The directory path under which a fileset will appear to the file system. The name of the directory is indicated by the DirectoryName property. The fully qualified directory name of this fileset is indicated by the AttachPoint property. This property is read-only.
Parent	string	The name of the parent fileset. If this is the global fileset, the value is null. Default is null. This property is read-only.
NumberOfChildren	uint32	The number of immediate child filesets. Default is 0. This property is read-only.
Quota	uint32	The maximum size limit, in megabytes, for the fileset. A value of zero, the default, indicates that there is no limit. The maximum value is 1 024 petabytes. This property is writable.
IsHardQuota	boolean	The indicator of whether a quota limit cannot be extended, which is a hard quota. This property is used when a quota limit exists and the fileset's allocated size reaches the quota limit. If the value is True, the server does not extend the allocated size of the fileset beyond the quota limit. It sends a Severe alert message and logs the message in the server message log. If False, the quota is soft. The server extends the allocated size of the fileset and logs a Warning alert message. This property is writable. Changing this property from soft quota (False) to hard quota (True) when the fileset has exceeded its quota causes a Hard Quota Violation (72) exception.
AlertPercentage	uint16	The percentage of the fileset size that, when reached, causes the server to generate an alert message. An alert is generated only if all the following conditions are met: • The Quota property value is greater than zero. • An AlertPercentage property value is greater than zero. • The SizeAllocatedPercentage property value equals or exceeds the AlertPercentage property value. This property is writable. Minimum is 0% and indicates that the server should not generate an alert. The maximum value is 100%. The default value is 80%.

Table 33. STC_Container class properties (continued)

Name	Type	Description
SizeAllocated	uint64	The size, in MB, of the fileset. This size can change as files are added and deleted in the fileset. This property is read-only.
SizeAllocatedPercentage	uint16	The percentage of the size allocated compared to the quota in the pool. This can be compared directly with the AlertPercentage to determine how close the fileset is to causing an alert. This property is read-only. The minimum value is 0%. The maximum value is 100%.
NumberOfPITCopies	uint16	The number of existing FlashCopy images. A fileset can have as many as 32 read-only FlashCopy images. When an administrator creates a FlashCopy image that causes the maximum number of images to be exceeded, SAN File System deletes the oldest existing image. This property is read-only.
LastPITCopyDate	datetime	The datetime value of the last FlashCopy image. This property is read-only.
AssignmentPolicy	uint32	The server assignment policy for this fileset. Possible values are: 0: Dynamic - SAN File System dynamically assigns the fileset to a server. 1: Static - The Administrator assigned the fileset to a specific server. If the assigned server stops, SAN File System will choose another server to host the fileset until the assigned server starts again. This property is read-only.
AssignedServer	string	The server name of the server to which the Administrator statically assigned the fileset. If the AssignmentPolicy property is Dynamic(0), this property will be null. This property is read-only.
Server	string	The name of the server currently hosting this fileset. This property is read-only. The maximum length is 256 characters.
ServerState	uint32	The state of the server serving the fileset. This property is read-only. Possible values are: 0: Offline 1: Online

Attach() method

Use the Attach() method to attach an existing fileset to a file system namespace. You can also use this method to reattach an attached fileset to a new file system namespace. When you reattach a fileset, you can change the directory path as well as the directory name.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 34 on page 100 describes the parameters that you can specify for the Attach() method.

Table 34. Attach() method parameters

Name	Type	Description
ExistingDirPath	string	Input parameter that is an existing directory path to attach to.
NewDirName	string	Input parameter that is the new directory name of the fileset.

Return values

The Attach() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 9 (Invalid name)
- 18 (The name given in the NewDirName parameter exists.)
- 21 (Directory path given in ExistingDirPath parameter not found)
- 22 (Not the primary Administrative server)
- 23 (Not viable; for reattach, the new path for the fileset to be attached already contains the original attach point name for the fileset.)
- 30 (Transaction failed)
- 36 (Is already attached)
- 44 (Incompatible operation)
- 61 (Cannot connect to server; the Administrative server could not contact the local Metadata server.)
- 62 (Too many connections)
- 65 (Server state offline)
- .. (Internal error)

ChangeServer() method

Use the ChangeServer() method to change the metadata server hosting the fileset.

You can change the server under the following conditions:

- The specified server acting as the new host must be part of the cluster.
- The cluster and the specified server must be either online or in a quiescent state.
- The current host server can be down. If it is not down, it and the cluster must be in a quiescent state.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 35 describes the parameters that you can specify for the ChangeServer() method.

Table 35. ChangeServer() method parameters

Name	Type	Description
Server	string	Input parameter that is the name of the metadata server to host this fileset. The maximum length is 32 characters.

Return values

The ChangeServer() method returns one of the following codes:

- 0 (Completed successfully)
- 3 (Already defined)
- 8 (Integrity lost)
- 10 (Invalid parameter)
- 21 (Not found)
- 22 (Not the primary administrative server)
- 27 (Is global fileset)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 45 (Server not found)
- 46 (Invalid cluster state)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 65 (Server state offline)
- .. (Internal error)

Create() method

Use the Create() method to define a new fileset. This method is the constructor for the class. Optionally, you can attach the fileset to an attach point by supplying an existing directory path and the new Directory name.

Execute Role: Administrator

Method Type: Static

Parameters

Table 36 describes the parameters that you can specify for the Create() method.

Table 36. Create() method parameters

Name	Type	Description
Name	string	Input parameter that is your label for the fileset. The maximum length is 256 characters.
Description	string	Input parameter that is your description of the fileset. The maximum length is 256 characters.
Quota	uint64	Input parameter that is the maximum size limit, in megabytes, for the fileset. A value of zero indicates that there is no limit. The default is no limit. The maximum value is 1024 petabytes.
IsHardQuota	boolean	Input parameter that is an indicator of whether a quota limit cannot be extended.
AlertPercentage	uint16	Input parameter that is the percentage of the fileset; size that, when reached, will cause the server to generate an alert message. The minimum value is 0% and indicates that the server should not generate an alert. The maximum value is 100%. The default value is 90%.
ExistingDirPath	string	Input parameter that is an existing directory path to attach to.

Table 36. Create() method parameters (continued)

Name	Type	Description
NewDirName	string	Input parameter that is the new directory name to be given to the fileset.
Server	string	Input parameter that is the name of the server to host this fileset. The maximum length is 32 characters.

Return values

The Create() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 10 (Invalid parameter)
- 18 (Name exists)
- 21 (Directory path given in ExistingDirPath parameter not found)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 43 (The name given in the NewDirName parameter exists.)
- 44 (Incompatible operation)
- 45 (Server not found)
- 57 (No space - The metadata server ran out of space in system volumes where master metadata is stored.
- 61 (Cannot connect to server - The current hosting server and all other parent hosting servers, if any, must be online.)
- 62 (Too many connections)
- 65 (Server state offline)
- .. (Internal error)

Delete() method

Use the Delete() method to delete a fileset.

You can delete a fileset under the following conditions:

- The fileset is detached.
- The fileset is not the global fileset.
- The fileset does not have files on it unless the IsForce option is set to True.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 37 describes the parameters you can specify for the Delete() method.

Table 37. Delete() method parameters

Name	Type	Description
IsForce	boolean	Input parameter that indicates whether to delete the fileset even if it has files on it.

Return values

The Delete method returns one of the following codes:

- 0 (Completed successfully)
- 5 (In use - Fileset has files and IsForce is False.)
- 8 (Integrity lost)
- 14 (Is referenced - Fileset is referenced in an active policy rule.)
- 21 (Not found)
- 22 (Not the primary administrative server)
- 27 (Is global fileset)
- 30 (Transaction failed)
- 30 (Is attached)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 65 (Server state offline; with or without the force option, the serving server must be online.)
- .. (Internal error)

Detach() method

Use the Detach() method to detach a fileset from a file system namespace.

You can detach a fileset under the following conditions:

- Fileset does not have any child filesets attached to it.
- No clients are using files on it unless the IsForce option is set to True.

When you detach a fileset, the attach point does not persist. The directory path and directory name are lost.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 38 describes the parameters that you can specify for the Detach() method.

Table 38. Detach() method parameters

Name	Type	Description
IsForce	boolean	Input parameter that indicates whether to detach the fileset even if clients are using files on it.

Return values

The Detach() method returns one of the following codes:

- 0 (Completed successfully)
- 5 (In use)
- 8 (Integrity lost)
- 14 (Is referenced; IsForce is False and clients are using files in the fileset)

- 20 (Not attached)
- 21 (Not found)
- 22 (Not the primary administrative server)
- 27 (Is global fileset)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 65 (Server state offline)
- .. (Internal error)

GetFileSetInfo() method

Use the GetFileSetInfo() method to retrieve fileset information for filesets in the cluster.

This method returns information about one or more filesets. It uses a set of array output parameters to represent information about each fileset.

Execute Role: Backup

Method Type: Static

Parameters

Table 39 describes the parameters that you can specify for the GetFileSetInfo() method.

Table 39. GetFileSetInfo() method parameters

Name	Type	Description
FileSetNames	string	Input parameter that is the list of fileset names. The maximum length is 256 characters.
FileSetName	string[ ]	Output parameter that is the name of the fileset.
ServerName	string[ ]	Output parameter that is the name of the metadata server serving this fileset.
CurrentTransactions	uint64[ ]	Output parameter that is the number of transactions accessing this fileset.
StoppedTransactions	uint64[ ]	Output parameter that is the number of transactions that failed to finish their work with this fileset.
RetriedTransactions	uint64[ ]	Output parameter that is the number of times some of the transactions have been retried. This number reports the number of deadlocks that are occurring.
StartedTransactions	uint64[ ]	Output parameter that is the number of transactions that used or are using this fileset.
CompletedTransactions	uint64[ ]	Output parameter that is the total number of transactions that successfully finished their work with this fileset.

Return values

The GetFileSetInfo() method returns one of the following codes:

- 0 (Completed successfully)

- 8 (Integrity lost)
- 21 (Not found; the fileset name was not found in the global namespace.)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- .. (Internal error)

ListAssociatedPools() method

Use the ListAssociatedPools() method to list the storage pools that a fileset can use to store data.

This method can return information about one or more storage pools. It uses a set of array output parameters to represent information about each storage pool.

Execute Role: Monitor

Method Type: Dynamic

Parameters

Table 40 describes the parameters that you can specify for the ListAssociatedPools() method.

Table 40. ListAssociatedPools() method parameters

Name	Type	Description
StgPoolName	string[ ]	Output parameter that is the name of the storage pool associated to this fileset.
InUse	boolean[ ]	Output parameter that indicates whether the named storage pool is currently used by this fileset.
PolicyRuleReferenceCount	uint32[ ]	Output parameter that is the number of policy rules that associate this storage pool with this fileset.

Return values

The ListAssociatedPools() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

Move() method

Use the Move() method to move or rename a fileset by creating a new fileset with the specified new name and migrating the data and capabilities to the new name. If successful, the old fileset is deleted.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 41 describes the parameters that you can specify for the Move() method.

Table 41. Move() method parameters

Name	Type	Description
NewName	string	Input parameter that is your new label for the fileset. The maximum length is 256 characters.

Return values

The Move() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 9 (Fileset name is not valid.)
- 18 (Fileset name already exists.)
- 21 (Fileset not found)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

RemoveServerBinding() method

Use the RemoveServerBinding() method to remove the static assignment that a fileset has to a metadata server. The fileset is immediately assigned a new server and its assignment policy becomes dynamic.

Execute Role: Administrator

Method Type: Dynamic

Return values

The RemoveServerBinding() method returns one of the following codes:

- 0 (Completed successfully)
- 21 (Not found)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 46 (Invalid cluster state)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

STC_LdapDynamicSetting

The STC_LdapDynamicSetting class contains the settings for LDAP configuration parameters that you can dynamically update without a cluster restart. These parameters persist across cluster restarts. If you have Administrator privileges, you can change the writable properties in this class using the SetProperty() intrinsic method. This class extends the STC_Setting class.

Properties

The STC_LdapDynamicSetting class has the following properties:

Table 42. STC_LdapDynamicSetting class properties

Name	Type	Description
ServerIP	string	The IP address of the LDAP server. This property is writable.
User	string	The root user for LDAP server for authentication. This property is writable.
Password	string	The root password for LDAP server for authentication. This property is writable.
SecuredConnection	boolean	An indicator of whether to use a Secure Sockets Layer (SSL) connection with LDAP. This property is writable.
BaseDnRoles	string	The base DN for role information. This property is writable.
RoleIdAttr	string	The attribute name of the role ID. This property is writable. The default value is ″cn″.
RoleMemIdAttr	string	The attribute name of the user reference. This property is writable. Default is ″roleOccupant″.
UserIdAttr	string	The attribute name of the user ID within reference. This property is writable. The default value is ″uid″.
CacheAge	uint32	The duration (in seconds) to hold LDAP entries in the LDAP cache. This property is writable. The default value is 600.

STC_MasterDisruptiveSetting

The STC_MasterDisruptiveSetting class represents the parameter settings for cluster configuration that require a cluster restart for an update to take effect. This class extends the STC_Setting class.

The read-only properties in this class can be set only during installation. If you have Administrator privileges, you can change the writable properties using the SetProperty() intrinsic method.

Properties

The STC_MasterDisruptiveSetting class has the following properties:

Table 43. STC_MasterDisruptiveSetting class properties

Name	Type	Description
ClusterID	uint32	A unique cluster ID number that is set only at installation time. This property is read-only. The default value is the lower 16-bit value of the system time during installation.

Table 43. STC_MasterDisruptiveSetting class properties (continued)

Name	Type	Description
ClusterName	string	A unique cluster name that is set only at installation time. This cluster name determines the name of the root directory in the file system namespace. This property is read-only. The maximum length is 32 characters.
ClientTimeoutInterval	uint32	The amount of time, in milliseconds, to wait between attempted message sends from server to client. This property is read-only. The minimum length of time is 200 and the maximum length of time is 1000 milliseconds. The default value is 500.
ServerTimeoutInterval	uint32	The amount of time, in milliseconds, to wait between attempted message sends from server to server. This property is read-only. The minimum length of time is 200 and the maximum length of time is 1000 milliseconds. The default value is 500.
DiskHeartbeatInterval	uint32	The interval, in milliseconds, between heartbeats written to disk. This property is read-only. The minimum length of time is 200 and The maximum length of time is 10 000 milliseconds. The default value is 500.
LogicalPartitionSize	uint32	The logical partition size in megabytes. This property is read-only. The default is 16 MB.
NWHeartbeatInterval	uint32	The interval, in milliseconds, between the heartbeats over the network. This property is writable. The minimum length of time is 200 and The maximum length of time is 10 000 milliseconds. The default value is 500.
NWMaxMissedHeartbeats	uint32	The maximum number of heartbeats that can be missed before the cluster ejects the server. If a server is ejected, it does not need to be recommissioned into the cluster. This property is writable. The minimum value is 1 and the maximum value is 100. The default value is 3.
DiskMaxMissedHeartbeats	uint32	The maximum number of heartbeats that can be missed before the disk ejects a server. If a server is ejected, it does not need to be recommissioned into the cluster. This property is writable. The minimum value is 1 and the maximum value is 100. The default value is 4.
LockLeasePeriod	uint32	The amount of time, in seconds, a lock is leased to a client when the server grants a lock. The server applies a multiplier, specified by the LockGracePeriodMultiplier property, before actually expiring the lease. This property is writable. The minimum value is 10 and the maximum value is 120. The default value is 20 seconds.
LockGracePeriodMultiplier	uint32	The value that the server would multiply times the lock lease period to determine the amount of time to wait before actually expiring a lease of a lock to a client. The server must receive a lock renewal request from the client during this time to keep the lease active. This property is writable. The minimum value is 0 and the maximum value is 4. The default value is 2 seconds.
ClusterTimeout	uint32	The timeout, in microseconds, for communications within the cluster. This property is writable. The minimum value is 500 000 and the maximum value is 10 000 000. The default value is 1 000 000.
RetriesToClient	uint32	The number of times a server attempts to send to a client before declaring the client dead. This property is writable. The minimum value is 1 and the maximum value is 100. The default value is 5.

Table 43. STC_MasterDisruptiveSetting class properties (continued)

Name	Type	Description
Lang	string	The language that the metadata server runs, which can be United States English or Japanese. Possible values are: en_US.utf8 ja_JP.utf8 The initial value is en_US.utf8.

GetInstalledLanguages() method

Use the GetInstalledLanguages() method to list the languages installed in SAN File System.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 44 describes the parameter for the GetInstalledLanguages() method.

Table 44. GetInstalledLanguages() method parameters

Name	Type	Description
Languages	string[ ]	Output parameter that is the list of installed languages.

Return values

The GetInstalledLanguages() method returns one of the following codes:

- 0 (Completed successfully)
- 4 (Command failed)
- 8 (Integrity lost)
- 30 (Transaction failed)
- .. (Internal error)

STC_MasterDynamicSetting

The STC_MasterDynamicSetting class represents the parameter settings for cluster configuration that you can dynamically update without a cluster restart. These parameters persist across cluster restarts. This class extends the STC_Setting class.

If you have Administrator privileges, you can change the writable properties in this class using the SetProperty() intrinsic method.

Properties

The STC_MasterDynamicSetting class has the following properties.

Table 45. STC_MasterDynamicSetting class properties

Name	Type	Description
MasterBufferSize	uint32	The buffer-cache size for master database space in 4 KB pages. This property is writable. The minimum value is 2 048 and the maximum value is 8 192 KB. The default value is 2 048 KB.
SubordinateBufferSize	uint32	The buffer-cache size for subordinate database space in 4 KB pages. This property is writable. The minimum value is 30 000 and the maximum value is 250 000 KB. The default value is 30 000 KB.
SpaceReclaimDelay	uint32	The interval, in minutes, that the space-reclamation thread waits between runs. A value of zero indicates that space reclamation is disabled. This property is writable. The minimum value is 0 and the maximum value is 1 440 minutes. The default value is 60 minutes.
PrivilegedFSClients	string	A comma-separated list of client names for whom administrator privileges are granted in the file system namespace. This property is writable.
SNMPEvents	uint16	A filter that decides if a Simple Network Management Protocol (SNMP) trap is to be generated when a significant event occurs in a server. You can choose the severity of the events that generate an SNMP trap by setting the corresponding bit in this property. This property is writable. Possible values are: • 0: Information • 1: Warning • 2: Error • 3: Severe If this value is set to zero (no bits set), SNMP trap generation is disabled. If all the bits are set to one, all event messages generate SNMP traps.
SNMPManagers	string	A comma-separated list of destination Internet Protocol (IP) addresses, in dotted decimal format, of the SNMP managers. If an SNMP trap is generated, the trap is sent to this list of managers. This property is writable.
NumAdminThreads	uint32	The number of threads for administrative operations. This value can only be increased and not decreased. This property is writable. The minimum value is 1 and the maximum value is 10. The default value is 4.
NumWorkerThreads	uint32	The number of threads for general operations. This value can only be increased and not decreased. This property is writable. The minimum value is 10 and the maximum value is 50. The default value is 10.

STC_MasterMetrics

The STC_MasterMetrics class represents the metrics for a cluster. Only one instance of this class should exist. This class extends the CIM_ServiceStatisticalInformation class.

The metrics include the current totals for the following types of buffers:

- Clean - Buffers that contain data but are available for reuse.
- Dirty - Buffers that contain data that is awaiting input/output (I/O) to disk.
- Free - Buffers that are available because they are currently not in use.

Properties

The STC_MasterMetrics class has the following properties:

Table 46. STC_MasterMetrics class properties

Name	Type	Description
TotalSystemMetaActivity	uint64	The total number of transactions relating to metadata activity for system objects. System objects include storage pools, filesets, volumes, policies, and engines. The activity includes read, create, delete, and modify operations on these objects. This counter property is read-only.
TotalSystemMetaUpdateActivity	uint64	The total number of transactions relating to metadata updates for system objects. This counter property is read-only.
TotalSystemBuffers	uint32	The current number of total buffers for system metadata activity. This property is read-only.
CleanSystemBuffers	uint32	The current number of clean buffers for system metadata activity. Clean buffers contain data but the buffers are available for reuse. This property is read-only.
DirtySystemBuffers	uint32	The current number of dirty buffers for system metadata activity. Dirty buffers contain data awaiting I/O to disk. This property is read-only.
FreeSystemBuffers	uint32	The current number of free buffers for system metadata activity. Free buffers are available because they are currently not in use. This property is read-only.

STC_MasterSAP

The STC_MasterSAP class represents the service access point of the master metadata server in a cluster. It extends the STC_TankSAP class.

STC_MasterService

The STC_MasterService class, along with the STC_Cluster class, provides cluster services. It extends the CIM_ClusteringService class.

Properties

The STC_MasterService class has the following properties:

Table 47. STC_MasterService class properties

Name	Type	Description
CurrentState	uint32	The state of the cluster. This property is read-only. Possible values are: 0: Down 1: Online 2: Partly Quiescent - Only metadata server I/O operations are suspended. 3: Fully Quiescent - All background I/O, client, and metadata server operations are suspended. 4: Administrative Quiescent - No longer servicing clients. 5: Forming a cluster 6: Not the master metadata server anymore 7: Unknown - Master metadata server could not be contacted to determine the cluster state. The probable reasons for this would be that the master is down or a network partition.
PendingState	uint32	The current state of the cluster transitions to this state if the current state is different from this pending state. This property is read-only. Possible values are the same as the CurrentState property.
LastCurrentStateChangeTime	datetime	The time passed since the cluster changed its current state. This property is read-only.
LastPendingStateChangeTime	datetime	The time passed since the cluster has had a state change pending. This property is read-only.
CommittedVersion	string	The committed software release version. This property is read-only.
CommittedUpgradeTimestamp	datetime	The timestamp when the latest upgrade was committed. This property is read-only.
CurrentVersion	string	The current software release version. This version will be different from the previous (committed) version if there was an upgrade done but the commit action was not yet activated. This property is read-only.
IsUpgradeInProgress	boolean	An indicator of whether an upgrade is in progress. After a version is committed, the system might take some time to sync up all the internal data structure versions. This property indicates if such a change is still in progress. This property is read-only.

AddServer() method

Use the AddServer() method to add a metadata server to the cluster.

Execute Role: Administrator

Method Type: Dynamic

Properties

Table 48 describes the parameters you can specify for the AddServer() method.

Table 48. AddServer() method parameters

Name	Type	Description
Ip	string	Input parameter that is the IP address of the Ethernet interface for the engine. The minimum value is 1024, and the maximum value is 65535.
ClusterPort	uint32	Input parameter that is the cluster port used by internal group services infrastructure communication. The default value is 1737.

Return values

The AddServer() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 4 (Command failed)
- 8 (Integrity lost)
- 12 (I/O failed)
- 16 (Message error)
- 18 (Already exists; another metadata server with the same name or the same IP and port values already exists.)
- 22 (Not the primary administrative server)
- 23 (Not viable; the server that you are adding is running on an engine that belongs to an incorrect subnetwork. Tip: The correct network is determined by the netmask value on the engine running the master metadata server.)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 74 (Values unequal; the configuration parameter values in the cluster do not match the values in the server that you are adding. Normally this is a transient problem because the server being rejected goes down after writing the new values in its local configuration file. If you restart the server and add the server again, this problem should not occur again. Note that if the server watchdog is enabled, the server will be restarted automatically.)
- .. (Internal error)

CommitUpgrade() method

Use the CommitUpgrade() method to obligate the cluster to start using an upgraded software version level. All the metadata servers in the cluster must first be upgraded to this version level.

Execute Role: Administrator

Method Type: Dynamic

Return values

The CommitUpgrade() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 68 (Already in progress; another commit is already in progress.)
- 69 (Up-to-date; the committed version is the same as the software version of all the servers.)
- 70 (All the servers in the cluster are not at the same software version.)
- .. (Internal error)

DropServer() method

Use the DropServer() method to stop and drop a metadata server from the cluster. This method requires that you specify the IP address and port number of the server that you want to drop from the cluster. If the server has filesets statically assigned to it, the filesets are moved to other servers.

Execute Role: Administrator

Method Type: Dynamic

Properties

Table 49 describes the parameters that you can specify for the DropServer() method.

Table 49. DropServer() method parameters

Name	Type	Description
Ip	string	Input parameter that is the IP address of the Ethernet interface for the engine. The minimum value is 1024, and the maximum value is 65535.
ClusterPort	uint32	Input parameter that is the cluster port used by internal group services infrastructure communication. The default value is 1737.

Return values

The DropServer() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 4 (Command failed)
- 5 (In use; there are some filesets statically assigned to the server that you are dropping.)
- 8 (Integrity lost)
- 16 (Message error)
- 21 (Not found)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)

- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

DropServerByName() method

Use the DropServer() method to stop and drop a metadata server from the cluster. This method requires that you specify the IP address and port number of the server that you want to drop from the cluster. If the server has filesets statically assigned to it, the filesets are moved to other servers.

Execute Role: Administrator

Method Type: Dynamic

Properties

Table 50 describes the parameters that you can specify for the DropServerByName() method.

Table 50. DropServerByName() method parameters

Name	Type	Description
ServerName	string	The input parameter that is the name of the metadata server that you want dropped.

Return values

The DropServerByName() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 4 (Command failed)
- 5 (In use)
- 8 (Integrity lost)
- 16 (Message error)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 45 (Server not found)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

FileSystemCheck() method

Use the FileSystemCheck() method to check and repair metadata.

This method provides options for you to:

- Check the integrity of the structure and the content of the metadata.
- Check the integrity of the system metadata and the user (fileset) metadata.
- Limit the user metadata checking to a subset of filesets.

You can restrict this operation to check only or check and repair. The message log contains a report generated by this method. If you did not limit the mode to check only, the system automatically salvages and repairs the damaged data if possible. Some types of repair require manual intervention from the administrator. In those cases, the cluster state is placed in Administrative mode. You can invoke the StopFileSystemCheck() method to stop a check and repair of metadata.

Note:

1. This method is a long-running process. If there is a cluster reformation while the method is running, this method might stop.
2. Only one FileSystemCheck() operation can be in progress at time.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 51 describes the parameters that you can specify for the FileSystemCheck() method.

Table 51. FileSystemCheck() method parameters

Name	Type	Description
IsCheckOnly	boolean	The input parameter that is an indicator of whether to only check and not repair.
CheckScope	uint16	The input parameter that is a bitmap indicating the scope of the check. Possible values are: 0: Structure - Checks the structure of the metadata. 1: Content - Checks the contents of the metadata. You can set both bits to check the structure and content.
Type	uint16	The input parameter that is a bitmap indicating the type of the metadata to be checked. Possible values are: 0: System - Checks the system metadata. 1: User - Checks the user (fileset) metadata. You can set both bits to check the system and user metadata.
ContainerList	string[ ]	The input parameter that is a list of filesets to be checked or repaired if the Type parameter is set to User and not System.

Return values

The FileSystemCheck() method returns one of the following codes:

- 0 (Completed successfully)
- 5 (In use; another FileSystemCheck operation is active.)
- 8 (Integrity lost; check-only option was chosen and the server detects corruption.)
- 10 (Invalid parameter)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 60 (Canceled; due to StopFileSystemCheck method.)

- 61 (Cannot connect to server)
- 62 (Too many connections)
- 73 (Salvaged; repair was requested. Corruption was detected and repaired successfully. This return code indicates success, not an error.)
- 76 (Cancel pending; aFileSystemCheck() method is active with a cancel pending. Make sure that the previous FileSystemCheck() method has stopped completely before issuing another check.)
- 77 (Salvage failed; a repair attempt failed. You can look in the server log files for details. Contact technical support and initiate metadata and data recovery actions.)
- .. (Internal error)

GetFileInfo() method

Use the GetFileInfo() method to retrieve the metadata information for a file.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 52 describes the parameters that you can specify for the GetFileInfo() method.

Table 52. GetFileInfo() method parameters

Name	Type	Description
FileName	string	This input/output parameter is the name of the file for which you are retrieving information.
FileType	uint32	This output parameter indicates the type of file. Possible values are: 1: File 2: Directory 3: SymbolicLink 4: Unknown
CreateTimeStamp	datetime	This output parameter is the time stamp for the file creation.
AccessTimeStamp	datetime	This output parameter is the time stamp for the last file access.
ModifyTimeStamp	datetime	This output parameter is the time stamp for the last modification of the data in the file.
AttrChangeTimeStamp	datetime	This output parameter is the time stamp for the last modification of the file attributes.
FileSize	uint64	This output parameter is the file size (in bytes).
BlockCount	uint64	This output parameter is the block file size (in bytes).
LinkCount	uint32	This output parameter is the number of names that reference this file object.
StoragePoolName	string	This output parameter is the name of the storage pool assigned for the file.
FileSetName	string	This output parameter is the name of the fileset to which the file belongs.

Table 52. GetFileInfo() method parameters (continued)

Name	Type	Description
ServerName	string	This output parameter is the name of the metadata server serving the file.

Return values

The GetFileInfo() method returns one of the following codes:

- 0 (Completed successfully)
- 10 (Invalid parameter)
- 21 (Not found)
- 22 (Not the primary administrative server)
- 44 (Incompatible operation)
- 65 (Server state offline)
- 86 (Operation unsupported by server)
- 87 (Not a directory)
- .. (Internal error)

ListClientsByLUN() method

Use the ListClientsByLUN() method to list the clients that have access to a specified LUN.

This method can return a list of one or more clients. It uses an array output parameters to list the clients.

Execute Role: Monitor

Method Type: Dynamic

Parameters

Table 53 describes the parameters for the ListClientsByLUN() method.

Table 53. ListClientsByLUN() method parameters

Name	Type	Description
LunID	string	The input parameter that is the identifier of the LUN.
ClientName	string[ ]	The output parameter that is the name of a client that can access the LUN.

Return values

The ListClientsByLUN() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)

- 89 (Invalid LUN ID)
- .. (Internal error)

ListClientsByVolume() method

Use the ListClientsByVolume() method to list the clients that have access to a specified volume.

This method can return a list of one or more clients. It uses an array output parameters to list the clients.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 54 describes the parameters for theListClientsByVolume() method.

Table 54. ListClientsByVolume() method parameters

Name	Type	Description
VolumeName	string	Input parameter that is the name of the volume.
ClientName	string[ ]	Output parameter that is the name of a client that can access the volume.

Return values

The ListClientsByVolume() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 79 (System volume)
- .. (Internal error)

QuiesceService() method

Use the QuiesceService() method to place the cluster in a quiescent state to perform some backup-and-restore and administrative operations.

All the servers that currently belong to the cluster are brought into a quiescent state. A cluster might temporarily leave the quiescent state if a server leaves or joins the cluster. You can use the ResumeService() method to return a cluster to a fully online state from a quiescent state.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 55 describes the parameters that you can specify for the QuiesceService() method.

Table 55. QuiesceService() method parameters

Name	Type	Description
Mode	uint32	This input parameter is the quiescent state. Possible values are: 0: Partly Quiescent - A limited quiescent mode that allows client file data activity to continue but prevents client metadata activity and new client connections. This state allows a backup with metadata integrity but might not preserve file data integrity. 1: Fully Quiescent - A full quiescent mode that suspends all client metadata activity and file data activity and terminates all client sessions. This state allows a backup with metadata and file data integrity. 2: Administrative Quiescent - Administrative operations that do not permit client activity can be performed safely.

Return values

The QuiesceService() method returns one of the following codes:

- 0 (Completed successfully.)
- 1 (Not supported)
- 4 (Command failed)
- 8 (Integrity lost)
- 10 (Invalid parameter)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

ResumeService() method

Use the ResumeService() method to return the cluster to a fully online state from the quiescent state.

Execute Role: Administrator

Method Type: Dynamic

Return values

The ResumeService() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 4 (Command failed)
- 8 (Integrity lost)

- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

StartService() method

Use the StartService() method to bring up all pre-commissioned metadata servers in an online state on all engines. This method starts the master metadata server, verifies that the master metadata server is online and then starts all subordinate servers.

Execute Role: Administrator

Method Type: Dynamic

Return values

The StartService() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 4 (Command failed)
- 5 (In use; the cluster is running already. The master metadata server is running.)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 24 (Server timed out; the metadata server in the cluster was launched successfully but failed to come online after a maximum wait period.)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 66 (Metadata server restart service state cannot continue; this is only a warning; the cluster is started successfully.)
- 78 (Aborted; one or more metadata servers, including the master, has aborted.)
- 91 (Cannot contact remote CIM agent to start a subordinate server)
- .. (Internal error)

StartServiceInAdmin() method

Use the StartServiceInAdmin() method to bring up all pre-commissioned metadata servers in an offline state on all engines.

Execute Role: Administrator

Method Type: Dynamic

Return values

The StartServiceInAdmin() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)

- 4 (Command failed)
- 5 (In use; the cluster is running already. The master metadata server is running.)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 24 (Server timed out; the metadata server in the cluster was launched successfully but failed to come online after a maximum wait period.)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 66 (Metadata server restart service state cannot continue; this is only a warning; the cluster is started successfully.)
- 78 (Aborted; one or more metadata servers, including the master, has aborted.)
- 91 (Cannot contact remote CIM agent)
- .. (Internal error)

StopFileSystemCheck() method

Use the StopFileSystemCheck() method to stop a FileSystemCheck() method that is in progress.

Execute Role: Administrator

Method Type: Dynamic

Return values

The StopFileSystemCheck() method returns one of the following codes:

- 0 (Completed successfully. - The current FileSystemCheck() method is marked for cancellation.)
- 8 (Integrity lost)
- 21 (Not Found)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 76 (Cancel pending - A cancel has been issued using StopFileSystemCheck, but the cancel is still pending. A check operation might not stop immediately after you invoke the StopFileSystemCheck() method.
- .. (Internal error)

StopService() method

Use the StopService() method to gracefully bring down metadata servers on all engines of a cluster.

Execute Role: Administrator

Method Type: Dynamic

Return values

The StopService() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 4 (Command failed)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 24 (Server timed out)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 66 (Metadata server restart service state cannot continue)
- 91 (Cannot contact remote CIM agent)
- .. (Internal error)

STC_MDSAuditLog

The STC_MDSAuditLog class represents the aggregated, audit log file for a metadata server. This class extends the STC_MessageLog class. It inherits methods from the STC_MessageLog class that enable you to traverse the log and retrieve a specified number of log records.

Properties

The STC_MDSAuditLog class has the following properties:

Table 56. STC_MDSAuditLog class properties

Name	Type	Description
BackupLogFile Name	string	The absolute path and name of the backup log file. This is consistent across all engines in the cluster.

STC_MDSEventLog

The STC_MDSEventLog class represents the event log file for a metadata server. This class extends the STC_MessageLog class.

Though the event log file is represented as a separate file, the event log is primarily a filter that groups all event records from the STC_MDSMessageLog.

Properties

The STC_MDSEventLog class has the following properties:

Table 57. STC_MDSEventLog class properties

Name	Type	Description
BackupLogFile Name	string	The absolute path and name of the backup log file. This is consistent across all engines in the cluster.

STC_MDSMessageLog

The STC_MDSMessageLog class represents the aggregated, message log file for a metadata server. This class extends the STC_MessageLog class. It inherits methods from the STC_MessageLog class that enable you to traverse the log and retrieve a specified number of log records.

Properties

The STC_MDSMessageLog class has the following properties:

Table 58. STC_MDSMessageLog class properties

Name	Type	Description
BackupLogFile Name	string	The absolute path and name of the backup log file. This is consistent across all engines in the cluster.

STC_MessageLog

The STC_MessageLog class represents log files that are present in SAN File System. This class extends the CIM_MessageLog class.

The STC_MessageLog class enhances the iterator methods of traversing the log file as defined in CIM_MessageLog class in the following ways:

- It has a PositionToLastRecord() method in addition to the PositionToFirstRecord() method.
- The GetNextRecords() and GetPreviousRecords() methods set the direction of traversal from a given iterator and return one or more log records using a set of array output parameters. The parameter values at a given index into this array constitute a log record.

Properties

The STC_MessageLog class has the following properties:

Table 59. STC_MessageLog class properties

Name	Type	Description
LogFileName	string	The absolute path and name of the log file. This is consistent across all engines in the cluster.

ClearLog() method

Use the ClearLog() method to clear a message or audit log of all entries.

Execute Role: Administrator

Method Type: Dynamic

Return values

The ClearLog() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported. Check that the Capabilities property defined in the parent class specifies that the message log can be cleared.)

- 4 (Command failed; the clear operation failed on the metadata server)
- 8 (Integrity lost)
- 30 (Transaction failed)
- 41 (Partial data; the clear operation partially cleared the log.)
- 61 (Cannot connect to metadata server)
- 62 (Too many connections)
- .. (Internal error)

GetNextRecords() method

You can use the GetNextRecords() method to retrieve a specified number of records from a message log, starting from the record indicated by the IterationIdentifier parameter.

After the method retrieves the records, it advances the IterationIdentifier parameter to the record after the last record returned. If the traversal reaches the last record in the file, the method returns an End of Iteration return code. Subsequent calls to this method might return new records if they have been written to the log.

This method can return one or more log records. It uses a set of array output parameters to represent the log records. The parameter values at a given index into this array constitute a log record. These parameter values provide information about the message as well as its content.

Execute Role: Monitor

Method Type: Dynamic

Parameters

Table 60 describes the parameters that you can specify for the GetNextRecords() method.

Table 60. GetNextRecords() method parameters

Name	Type	Description
IterationIdentifier	string	Input/Output parameter that is an identifier for the iterator. The maximum length is 100 characters.
NumberOfEntries	uint32	Input/Output parameter that indicates the number of records to be retrieved and returns the actual number of log records that were retrieved.
MessageTimestamp	datetime[]	Output parameter with the timestamp for the message.
MessageID	string[]	Output parameter with the identifier for the message.
MessageType	uint8[]	Output parameter with the type of message. Possible values are: • 1: Normal • 2: Event • 3: Audit • 4: Trace
SourceNode	string[]	Output parameter with the identifier of the engine that originated the message.

Table 60. GetNextRecords() method parameters (continued)

Name	Type	Description
Severity	uint8[]	Output parameter with the severity of the message. Possible values are: • 0: Information • 1: Warning • 2: Error • 3: Severe
MessageString	string[]	Output parameter with the content of the message.

Return values

The GetNextRecords() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 8 (Integrity lost)
- 10 (Invalid parameter)
- 30 (Transaction failed)
- 37 (End of iteration)
- 38 (Invalid IterationIdentifier)
- 39 (File not found)
- 40 (Cannot read file)
- 41 (Partial data)
- .. (Internal error)

GetPreviousRecords() method

Use the GetPreviousRecords() method to retrieve a specified number of records from a message log, ending at the record indicated by the IterationIdentifier parameter.

After the method retrieves the records, it positions the IterationIdentifier parameter to the record before the first record returned. If the traversal reaches the first record in the file, the method returns an End of Iteration return code. Subsequent calls to this method will have no effect.

This method can return one or more log records. It uses a set of array output parameters to represent the log records. The parameter values at a given index into this array constitute a log record. These parameter values provide information about the message as well as its content.

Execute Role: Monitor

Method Type: Dynamic

Return values

Table 61 on page 127 describes the parameters that you can specify for the GetPreviousRecords() method:

Table 61. GetPreviousRecords() method parameters

Name	Type	Description
IterationIdentifier	string	Output parameter that is an identifier for the iterator. The maximum length is 100 characters.
NumberOfEntries	uint32	Input/Output parameter that indicates the number of records to be retrieved and returns the actual number of log records that were retrieved.
MessageTimestamp	datetime[]	Output parameter with the timestamp for the message.
MessageID	string[]	Output parameter with the identifier for the message.
MessageType	uint8[]	Output parameter with the type of message. Possible values are: • 1: Normal • 2: Event • 3: Audit • 4: Trace
SourceNode	string[]	Output parameter with the identifier of the engine that originated the message.
Severity	uint8[]	Output parameter with the severity of the message. Possible values are: • 0: Information • 1: Warning • 2: Error • 3: Severe
MessageString	string[]	Output parameter with the content of the message.

Return values

The GetPreviousRecords() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 8 (Integrity lost)
- 10 (Invalid parameter)
- 30 (Transaction failed)
- 37 (End of iteration)
- 38 (Invalid IterationIdentifier)
- 39 (File not found)
- 40 (Cannot read file)
- 41 (Partial data)
- .. (Internal error)

PositionToFirstRecord() method

Use the PositionToFirstRecord() method to establish an iteration of a message log and set the iterator to the first entry in the log. An identifier for the iterator is returned as an output parameter.

Execute Role: Monitor

Method Type: Dynamic

Parameters

Table 62 describes the parameters that you can specify for the PositionToFirstRecord() method.

Table 62. PositionToFirstRecord() method parameters

Name	Type	Description
IterationIdentifier	string	The output parameter that is an identifier for the iterator.

Return values

The PositionToFirstRecord() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 8 (Integrity lost)
- 30 (Transaction failed)
- 39 (File not found)
- 40 (Cannot read file)
- 41 (Partial data)
- .. (Internal error)

PositionToLastRecord method

Use the PositionToLastRecord() method to establish an iteration of a message log and set the iterator to the last entry in the log. An identifier for the iterator is returned as an output parameter.

Execute Role: Monitor

Method Type: Dynamic

Parameters

Table 63 describes the parameters that you can specify for the PositionToLastRecord() method.

Table 63. PositionToLastRecord() method parameters

Name	Type	Description
IterationIdentifier	string	The output parameter that is an identifier for the iterator.

Return values

The PositionToLastRecord() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 8 (Integrity lost)
- 30 (Transaction failed)
- 39 (File not found)
- 40 (Cannot read file)

- 41 (Partial data)
- .. (Internal error)

PositionWithFilter() method

Use the PositionWithFilter() method to establish an iteration of a message log at a specified date and with a severity filter. An identifier for the iterator is returned as an output parameter.

You can decide to position the iterator at either the beginning or end of a set of log entries that have the specified date. The severity filter enables you to specify that the iterator only return entries of a specified severity.

Execute Role: Monitor

Method Type: Dynamic

Parameters

Table 64 describes the parameters that you can specify for the PositionWithFilter() method.

Table 64. PositionWithFilter() method parameters

Name	Type	Description
IterationIdentifier	string	The output parameter that is an identifier for the iterator.
IsBeginning	boolean	The input parameter that indicates whether the position of the iterator is at the beginning of the specified date. If a date is not specified this parameter indicates whether the position of the iterator is at the beginning of the log.
DateFilter	datetime	The input parameter that specifies the date at which the iterator will be positioned. If the date is within the range of entries in the log, the iterator is positioned at the previous or next date based on the direction indicated by the IsBeginning parameter. If a specified date is outside the range in the log or is all zeroes, the iterator is positioned at the beginning or the end of the log, based on the IsBeginning parameter. Only the yyyymmdd fields in the datetime value are used.
SeverityFilter	uint8	The input parameter that specifies the severities of log entries that the iterator returns. Possible values are: 0: Information 1: Warning 2: Error 3: Severe The minimum value is 1 and the maximum value is 15.

Return values

The PositionWithFilter() method returns one of the following codes:

- 0 (Completed successfully)
- 1 (Not supported)
- 39 (File not found)
- 40 (Cannot read file)

- 41 (Partial data)
- .. (Internal error)

STC_NodeFan

The STC_NodeFan class represents status of a engine's fan. This class extends the CIM_LogicalElement class.

Properties

The STC_NodeFan class has the following properties:

Table 65. STC_NodeFan class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_ComputerSystem. This property is key. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_NodeFan. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
DeviceID	string.	An address or other identifying information to uniquely name the specific fan. This property is key. The maximum length is 64 characters.
Speed	uint32	Speed of the fan in percentage of the optimal speed of 100%.

STC_NodeTemperature

The STC_NodeTemperature class represents the temperature state of hardware components of an engine, as reported by the Advanced System Management Processor. An instance of this class exists for each temperature sensor available on every engine of a cluster. This class extends the CIM_LogicalElement class.

Properties

The STC_NodeTemperature class has the following properties:

Table 66. STC_NodeTemperature class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_ComputerSystem. This property is key. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_NodeTemperature. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
DeviceID	string	An address or other identifying information to uniquely name the temperature sensor, for example, CPU temperature. This property is key. The maximum length is 64 characters.

Table 66. STC_NodeTemperature class properties (continued)

Name	Type	Description
Value	real32	Current temperature, in degrees Celsius, of this hardware component. This property is read-only.
HasThresholds	boolean	Indicator of whether thresholds are available for this hardware component. This property is read-only.
WarningReset	real32	Temperature threshold value for warning reset. If the temperature exceeds the Warning property value and then drops below this value, the Advanced System Management Processor clears any active temperature events. A value of zero means that this threshold is disabled. This property is read-only.
Warning	real32	Temperature threshold value for warning. If the temperature reaches this value, the Advanced System Management Processor generates a warning event. A value of zero means that this threshold is disabled. This property is read-only.
SoftShutdown	real32	Temperature threshold value for soft shutdown. If the temperature reaches this value, a critical event is generated and the server is powered off after the operating system is shut down. A value of zero means that this threshold is disabled. This property is read-only.
HardShutdown	real32	Temperature threshold value for hard shutdown. If the temperature reaches this value, a critical event is generated and the server is powered off immediately. A value of zero means that this threshold is disabled. This property is read-only.

STC_NodeVitalProductData

The STC_NodeVitalProductData class represents vital product data about the components of a engine. This class extends the CIM_LogicalElement class.

Properties

The STC_NodeVitalProductData class has the following properties:

Table 67. STC_NodeVitalProductData class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_ComputerSystem. This property is key. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_NodeVitalProductData. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
DeviceID	string	An address or other identifying information to uniquely name the logical device. This property is key. The maximum length is 64 characters.
MachineModel	string	The model identifier of the logical device's host machine. This property is read-only.
SerialNumber	string	The serial number of the logical device's host machine. This property is read-only.

Table 67. STC_NodeVitalProductData class properties (continued)

Name	Type	Description
Revision	string	The firmware revision of the logical device. This property is read-only.
RevisionDate	datetime	The firmware revision date of the logical device. This property is read-only.
FirmwareFileName	string	The firmware file name of the logical device. This property is read-only.
FirmwareBuildID	string	The firmware build ID of the logical device. This property is read-only.

STC_NodeVoltage

The STC_NodeVoltage class represents the state of the voltage sources of a engine, as reported by the Advanced System Management Processor. There is an instance of this class for each voltage source available on every engine of a cluster. This class extends the CIM_LogicalElement class.

Properties

The STC_NodeVoltage class has the following properties:

Table 68. STC_NodeVoltage class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_ComputerSystem. This property is key. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_NodeVoltage. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
DeviceID	string	An address or other identifying information to uniquely name the logical device. This property is key. The maximum length is 64 characters.
CurrentVoltage	real32	The current voltage of a voltage source line on this engine.
DefaultVoltage	real32	The default voltage of a voltage source line on this engine.
HasThresholds	boolean	An indicator of whether thresholds are available for this device.
WarningLow	real32	The low value for a warning on this voltage line. If the voltage drops below this value, a warning event is generated. A value of -99.99 means that this threshold is disabled.
WarningHigh	real32	The high value for a warning on this voltage line. If the voltage rises above this value, a warning event is generated. A value of -99.99 means that this threshold is disabled.
SoftShutdownLow	real32	The low value for a soft shutdown of this voltage line. If the voltage drops below this value, a critical event is generated and the server is powered off after the operating system is shut down. A value of -99.99 means that this threshold is disabled.

Table 68. STC_NodeVoltage class properties (continued)

Name	Type	Description
SoftShutdownHigh	real32	The high value for a soft shutdown of this voltage line. If the voltage rises above this value, a critical event is generated and the server is powered off after the operating system is shut down. A value of -99.99 means that this threshold is disabled.
HardShutdownLow	real32	The low value for a hard shutdown of this voltage line. If the voltage drops below this value, a critical event is generated and the server is powered off immediately. A value of -99.99 means that this threshold is disabled.
HardShutdownHigh	real32	The high value for a hard shutdown of this voltage line. If the voltage rises above this value, a critical event is generated and the server is powered off immediately. A value of -99.99 means that this threshold is disabled.

STC_NodeWatchdog

The STC_NodeWatchdog class represents the settings for an Advanced System Management Processor watchdog. There is an instance of this class for each engine in the cluster. This class extends the CIM_LogicalElement class.

Properties

The STC_NodeWatchdog class has the following properties:

Table 69. STC_NodeWatchdog class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_ComputerSystem. This property is key. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_NodeWatchdog. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
Name	string	An address or other identifying information to uniquely name the watchdog. This property is key. The maximum length is 64 characters.
POSTTimeout	uint32	The watchdog timeout value for the Power-On Self Test (POST). The POST watchdog is active once when the power is coming up. If the engine fails to complete POST within the time indicated by this timeout value, the Advanced System Management Processor generates a POST timeout alert and automatically restarts the system once. When the system restarts, the POST watchdog is automatically disabled until the operating system is shut down and the server is power cycled. A value of zero indicates that this watchdog is disabled.
OSMonitorInterval	uint32	The frequency, in seconds, that the Advanced System Management Processor checks that the operating system is running properly. The operating system watchdog checks the state of the operating system at periodic intervals of time. A value of zero indicates that this watchdog is disabled.

Table 69. STC_NodeWatchdog class properties (continued)

Name	Type	Description
OSTimeout	uint32	The watchdog timeout value, in seconds, for the operating system. If the operating system fails to respond to these checks within this timeout value, the Advanced System Management Processor generates an operating system Timeout alert and automatically restarts the system. When the operating system restarts, the operating system watchdog is automatically disabled until the operating system is shut down and the server is power cycled.
LoaderTimeout	uint32	The watchdog timeout value, in seconds, for the operating system boot process or loader. The timeout value indicates the amount of time the Advanced System Management Processor waits between the completion of POST and the end of loading the operating system. If the interval is exceeded, the Advanced System Management Processor generates a Loader Timeout alert and automatically restarts the system once. When the system restarts, the Loader Timeout is disabled until the operating system is shut down and the server is power cycled. A value of zero indicates that this watchdog is disabled.
PowerOffDelay	uint32	The amount of time, in seconds, the Advanced System Management Processor waits for the operating system to shut down before powering off the system.

STC_PitImage

The STC_PitImage class represents the FlashCopy images (also known as point-in-time images) of a fileset. This class extends the CIM_ManagedSystemElement class.

There is an instance of this class for every FlashCopy images of the filesets that exists in a SAN File System. The space used by a FlashCopy image is accounted for in the space used by a container for quota calculations. When a FlashCopy image is created, it does not use any space. FlashCopy images use space when files within the fileset are modified after a FlashCopy image is taken.

Properties

The STC_PitImage class has the following properties:

Table 70. STC_PitImage class properties

Name	Type	Description
Caption	string	A one-line description of the object. This property is read-only. The maximum length is 64 characters.
ContainerName	string	Your label for the fileset to which this FlashCopy image belongs. This property is key. The maximum length is 256 characters.
Name	string	Your administrative name for the FlashCopy image. This property is key. The maximum length is 256 characters.
Description	string	Your description of the fileset. The maximum length is 256 characters.
InstallDate	datetime	The time when the FlashCopy image was created. A lack of a value does not indicate that the fileset does not exist. The alias is CreationDate.

Table 70. STC_PitImage class properties (continued)

Name	Type	Description
DirectoryName	string	The directory name containing this FlashCopy image. The full path for the FlashCopy image in the file system is given by AttachPoint/.pit/DirectoryName where AttachPoint is the attach point of the fileset.
State	uint32	The state of the FlashCopy image. Possible values are: 0: Complete 1: Incomplete

Create() method

Use the Create() method to create a new FlashCopy image for a fileset. This method is the constructor for this class.

When you create a FlashCopy image, the fileset can be attached or detached. Thirty-two FlashCopy images can exist at any given time. When this limit is reached, a Create operation fails unless the IsForce parameter is set to True. In this case, the oldest FlashCopy image is deleted so the new one can be created.

Execute Role: Backup

Method Type: Static

Parameters

Table 71 describes the parameters you can specify for the Create() method.

Table 71. Create() method parameters

Name	Type	Description
ContainerName	string	Input parameter that is your label for the fileset to which this FlashCopy image belongs. The maximum length is 256 characters.
Name	string	Input parameter that is your administrative name for the FlashCopy image. The maximum length is 256 characters.
Description	string	Input parameter that is your description of the fileset. The maximum length is 256 characters.
DirectoryName	string	Input parameter that is the new directory name to be given to the FlashCopy image.
IsForce	boolean	Input parameter that indicates whether to delete the oldest FlashCopy image copy to create this one when the limit is reached.

Return values

The Create() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 10 (Invalid parameter; the FlashCopy image name length is greater than the maximum, or Description length is greater than the maximum, or DirectoryName length is greater than the maximum, or the DirectoryName contains directory separators.)

- 18 (FlashCopy image name already exists for the fileset.)
- 21 (Fileset not found or no server serving the fileset.)
- 22 (Not the primary administrative server)
- 30 (Transaction failed; other concurrent activity in the server caused this create operation to fail.)
- 42 (Table full; the number of FlashCopy images taken are at the maximum limit already.)
- 43 (Directory name already exists for another FlashCopy image for the same fileset.)
- 44 (Incompatible operation; the server is executing an incompatible operation to this Create.)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 65 (Server state offline; any operation on a FlashCopy image needs the server serving the fileset to be online.)
- .. (Internal error)

Delete() method

Use the Delete method to delete a FlashCopy image of a fileset.

You cannot delete a FlashCopy image that has client activity (session locks open) unless the IsForce parameter is set to True. If the IsForce parameter is set to True, all the client activity is terminated (session locks revoked) before the delete.

Execute Role: Backup

Method Type: Dynamic

Parameters

Table 72 describes the parameters that you can specify for the Delete() method.

Table 72. Delete method parameters

Name	Type	Description
IsForce	boolean	Input parameter that indicates whether to delete the fileset even if client activity exists.

Return values

The Delete() method returns one of the following codes:

- 0 (Completed successfully)
- 5 (In use; client session locks are open and IsForce option is False.)
- 8 (Integrity lost)
- 21 (Not found; FlashCopy image not found or fileset not found, or no server serving the fileset.)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)

- 65 (Server state offline; any operation on a FlashCopy image needs the server serving the fileset to be online.)
- .. (Internal error)

Revert() method

Use the Revert() method to revert a fileset to this instance of the FlashCopy image.

You cannot revert a fileset that has children filesets. Detach children filesets, if any, manually.

A Revert operation deletes all the FlashCopy images that are more recent than this instance, including the current fileset image. If the IsForce parameter is False, you cannot revert a fileset to this instance under any of the following conditions:

- Any client activity exists (session locks open) in the FlashCopy images (including the current fileset) to be deleted.
- Any client activity exists (session locks open) in the current PIT instance.

In other words, the client activity can continue only in FlashCopy images taken earlier than this instance.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 73 describes the parameters you can specify for the Revert() method:

Table 73. Revert() method parameters

Name	Type	Description
IsForce	boolean	Input parameter that indicates whether to revert a fileset to this instance even if client activity exists.

Return values

The Revert() method returns one of the following codes:

- 0 (Completed successfully)
- 5 (In use; client session locks are open and IsForce option is False.)
- 8 (Integrity lost)
- 21 (Not found; FlashCopy image not found or fileset not found, or no server serving the fileset.)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 36 (Is attached; fileset has child filesets.)
- 41 (Partial data; FlashCopy image contains incomplete files so the fileset is not reverted.)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 65 (Server state offline; any operation on a FlashCopy image needs the server serving the fileset to be online.)

- .. (Internal error)

STC_PolicySet

The STC_PolicySet class represents a policy, which is a list of file-placement and service-class rules that define characteristics and placement of files. It extends the CIM_PolicySet class.

There is an instance of this class for every policy that exists in a SAN File System. Although multiple policies can exist in the system, only one policy can be active. The system defines a default policy set that assigns files to the Default storage pool.

Properties

The STC_PolicySet class has the following properties:

Table 74. STC_PolicySet class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_Cluster. This property is key. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_PolicySet. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
Name	string	A label for this policy. This property is key. The maximum length is 256 characters.
State	uint16	An indication whether or not this policy is administratively active. Only one policy can be active at any time. Possible values are: 0: Not Active 1: Active The default value is 0.
PolicyRules	string	The set of policy rules belonging to this policy.
Description	string	Your description of this policy. The maximum length is 256 characters.
CreationDate	datetime	The date and time when this policy was created.
LastModificationDate	datetime	The date and time when the rules in this policy were last modified. If the rules were never modified, this value will be the same as the creation date. The policy rules can be modified as a whole by creating a policy with the same name and using the IsForce option.
LastActiveDate	datetime	The date and time when this policy was last active. This is actually the date and time when another policy was made active (enabled) instead of this one. If the policy was never activated or is currently active, the value is null.

Activate() method

Use the Activate() method to activate a stored policy.

Execute Role: Administrator

Method Type: Dynamic

Properties

Table 75 describes the parameters you can specify for the Activate() method.

Table 75. Activate() method parameters

Name	Type	Description
Errors	string[]	Output parameter that contains information about a policy bind error if one occurred.

Return values

The Activate() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 21 (Not found)
- 22 (Not the primary administrative server)
- 25 (Policy bind errors; see the Errors output parameter for more information about this error.)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

Create() method

Use the Create() method to create a new policy. This method is the constructor for the class.

Execute Role: Administrator

Method Type: Static

Parameters

Table 76 describes the parameters that you can specify for the Create() method

Table 76. Create() method parameters

Name	Type	Description
Name	string	Input parameter that is a label for this policy. The maximum length is 256 characters.
Description	string	Input parameter that is your description of this policy. The maximum length is 256 characters.

Table 76. Create() method parameters (continued)

Name	Type	Description
PolicyRules	string	Input parameter that is the set of policy rules belonging to this policy. See the "File placement policy syntax" section in the *Administrator's Guide and Reference* for a description of the syntax conventions for policy rules.
IsForce	boolean	Input parameter that indicates whether an existing policy with the same name will be overwritten by this policy.
Errors	string []	Output parameter that contains information about a policy syntax error if one occurred.

Return values

The Create() method returns one of the following codes:

- 0 (Completed successfully)
- 3 (Already defined; another policy with the same name exists and the IsForce flag is False.)
- 5 (In use)
- 8 (Integrity lost)
- 9 (Invalid name; the name has invalid characters)
- 13 (Is default; the name is DEFAULT_POLICY and the IsForce flag is True. Cannot overwrite the default policy.)
- 21 (Not found)
- 22 (Not the primary administrative server)
- 26 (Policy syntax error; see the Errors output parameter for more information about this error including its location.)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

Delete() method

Use the Delete() method to delete an existing stored policy. The policy must be inactive for delete to succeed.

Execute Role: Administrator

Method Type: Dynamic

Return values

The Delete() method returns one of the following codes:

- 0 (Completed successfully)
- 5 (In use)
- 8 (Integrity lost)
- 13 (Is default; the name is DEFAULT_POLICY and the IsForce flag is True. Cannot delete default policy.)
- 21 (Not found)

- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

GetPolicyRuleStats() method

Use the GetPolicyRuleStats() method to retrieve the policy rule statistics associated with a specified fileset.

This method returns statistics associated with one or more specified filesets. It uses a set of array output parameters to represent the information for each fileset.

Execute Role: Monitor

Method Type: Static

Parameters

Table 77 describes the parameters that you can specify for the GetPolicyRuleStats() method:

Table 77. GetPolicyRuleStats() method parameters

Name	Type	Description
FileSetNames	string	Input parameter that is the list of fileset names.
Name	string[ ]	Output parameter that is the name of the rule.
Index	uint32[ ]	Output parameter that is the ordinal position of the rule.
Pool	string[ ]	Output parameter that is the name of the storage pool.
Last	datetime[ ]	Output parameter that is the time when a rule was last applied.
Hit	uint32[ ]	Output parameter that is the number of times that the rule has been evaluated and its conditions were true, so that the rule was applied.
Miss	uint32[ ]	Output parameter that is the number of times that the rule has been evaluated and its conditions were false, so that the rule was not applied.
Err	uint32[ ]	Output parameter that is the number of times that the rule has caused an error while being evaluated. These are not syntax errors.

Return values

The GetPolicyRuleStats() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 61 (Cannot connect to server)
- 62 (Too many connections)

- .. (Internal error)

GetRules() method

Use the GetRules() method to retrieve the set of rules associated with this policy.

Execute Role: Monitor

Method Type: Dynamic

Parameters

Table 78 describes the parameters that you can specify for the GetRules() method.

Table 78. GetRules() method parameters

Name	Type	Description
RulesList	string	This output parameter is the set of policy rules belonging to this policy. See the ″File placement policy syntax″ section in the *Administrator's Guide and Reference* for a description of the syntax conventions for policy rules.

Return values

The GetRules() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 21 (Not found)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

GetStoragePoolStats() method

Use the GetStoragePoolStats() method to retrieve the policy storage pool statistics associated with a specified fileset.

This method returns statistics associated with one or more specified filesets. It uses a set of array output parameters to represent the information for each fileset.

Execute Role: Monitor

Method Type: Static

Parameters

Table 79 on page 143 describes the parameters that you can specify for the GetStoragePoolStats() method.

Table 79. GetStoragePoolStats() method parameters

Name	Type	Description
FileSetNames	string	Input parameter that is the list of fileset names.
PoolName	string[ ]	Output parameter that is the name of the storage pool.
Count	uint32[ ]	Output parameter that is the number of times a file was placed into this storage pool.
Last	datetime[ ]	Output parameter that is the time a file was last placed into this storage pool.

Return values

The GetStoragePoolStats() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

STC_RegisteredFSClients

The STC_RegisteredFSClients class represents the registered clients of a metadata server. This class extends the CIM_LogicalElement class.

Every client-server registration pair is unique. The same client that registers to two servers appears as two instances.

Properties

The STC_RegisteredFSClients class has the following properties:

Table 80. STC_RegisteredFSClients class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_ComputerSystem. This property is key. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
ServiceCreationClassName	string	The class name of the scoping service: STC_TankService. This property is key. The maximum length is 256 characters.
ServiceName	string	The instance name of the scoping service. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_RegisteredFSClients. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
Name	string	The name of the client. The maximum length is 256 characters.
Id	uint64	The ID of the client.

Table 80. STC_RegisteredFSClients class properties (continued)

Name	Type	Description
IPAddress	string	The IP network address of the client.
IPPort	uint32	The IP network port address of the client.
Platform	string	The operating system platform of the client.
Version	string	The SAN File System version of the client.
LeaseRenewals	uint64	The total number of times the client has renewed a lease. This property indicates a measure of how long the client was active.
State	uint16	The state of the client's lease. Possible values are: • 0: Expired lease • 1: Valid lease
IsPrivileged	boolean	An indicator of whether this SAN File System client has Administrator privileges to the SAN File System namespace.
LastLeaseTimeStamp	datetime	The timestamp when the server issued or extended the lease.
ResidualLeaseTime	uint32	The countdown timer, in seconds, indicating how long the current lease will last. The lease time is two times the lease renewal interval, a configurable parameter, beginning from the LastLeaseTimeStamp property value.
Transactions	uint64	The total number of transactions started by this client.
CompletedTransactions	uint64	The total number of transactions completed.
SessionLocks	uint32	The current number of session locks this client is holding.
DataLocks	uint32	The current number of data locks this client is holding.
ByteRangeLocks	uint32	The current number of byte range locks this client is holding.

STC_RemoteServiceAccessPoint

The STC_RemoteServiceAccessPoint class represents a remote service access point. It provides information that you can use to access the SAN File System console. This class extends the CIM_RemoteServiceAccessPoint class.

STC_RsaDynamicSetting

The STC_RsaDynamicSetting class contains the settings for RSA configuration parameters that you can dynamically update, without a cluster restart. These parameters persist across cluster restarts. If you have Administrator privileges, you can change the writable properties in this class using the SetProperty() intrinsic method. This class extends the STC_Setting class.

Properties

The STC_RsaDynamicSetting class has the following properties:

Table 81. STC_RsaDynamicSetting class properties

Name	Type	Description
RsaUser	string	The user name for the RSA cards found on the RS-485 bus. The local RSA card is accessed directly. This property is writable.

Table 81. STC_RsaDynamicSetting class properties (continued)

Name	Type	Description
RsaPassword	string	The password for the RSA cards found on the RS-485 bus. The local RSA card is accessed directly. This property is writable.

STC_Setting

The STC_Setting class is the base class for cluster and server configuration parameters. This class extends the CIM_Setting class.

Properties

The STC_Setting class has the following properties:

Table 82. STC_Setting class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system. For the STC_MasterDisruptiveSetting and STC_MasterDynamicSetting classes, the class name of the scoping system is STC_Cluster. For the STC_TankDisruptiveSetting and STC_TankTransientSetting classes, the class name of the scoping system is STC_ComputerSystem. This property is key. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
ServiceCreationClassName	string	The class name of the scoping service. For the STC_MasterDisruptiveSetting and STC_MasterDynamicSetting classes, the class name of the scoping service is STC_MasterService. For the STC_TankDisruptiveSetting and STC_TankTransientSetting classes, the class name of the scoping service is STC_TankService. This property is key. The maximum length is 256 characters.
ServiceName	string	The instance name of the scoping service. This property is key. The maximum length is 64 characters.

STC_StoragePool

The STC_StoragePool class represents a storage pool. It extends the CIM_ManagedSystemElement class.

There is an instance of this class for every storage pool that exists in a SAN File System. The instances include a system storage pool that the SAN File System uses to maintain system metadata. At least one other storage pool must exist for client files, which is called the default storage pool. The list of instances and the methods defined in this class are available only on the master metadata server.

Properties

The STC_StoragePool class has the following properties:

Table 83. STC_StoragePool class properties

Name	Type	Description
Caption	string	A one-line description of the object. This property is read-only. The maximum length is 64 characters.
Name	string	Your label for the storage pool. This property is key. The maximum length is 256 characters.
PoolType	uint32	The type of pool. This property is read-only. Possible values are: 0: User 1: User Default 2: System The default is User. You can use the "SetDefault() method" on page 149 to change the storage pool type to User Default if the storage pool type is User.
PartitionSize	uint64	The partition size, in megabytes, to use when a fileset allocates space. This property is read-only. Possible values are 16, 64, and 256 MB. The default value is 16 MB.
AllocSize	uint32	The allocation strategy to use for files on this storage pool: • System allocation - An escalation algorithm is used to allocate blocks to a file placed on this storage pool. Indicated by a AllocSize value of zero. This is the default value. • Fixed allocation - The file is extended by a chosen fixed size every time. The fixed allocation size is indicated by this AllocSize value, either 4 KB or 128 KB. This property is read-only. Possible values are 0, 4, or 128. Default is 0.
AlertPercentage	uint16	The percentage of the estimated storage pool size that, when reached, causes the server to generate an alert message. This property is writable. The minimum percentage is 0% and indicates that the server should not generate an alert. The maximum percentage is 100%. The default value is 80%.
Size	uint64	The size, in megabytes, of the storage pool. The size of the storage pool is the sum of the sizes of the volumes within the storage pool. The size of the storage pool can change as volumes are added and deleted. This property is read-only.
SizeAllocated	uint64	The size, in megabytes, of the storage pool allocated to filesets. The files within a fileset use a portion of the size allocated to a fileset. The rest is free size. This property is the sum of the allocated sizes of the volumes within the storage pool. It is read-only.
SizeAllocatedPercentage	uint16	The percentage of the size allocated in the storage pool. You can compare this value to the AlertPercentage property value to determine how close the storage pool is to causing an alert. This property is read-only. The minimum percentage is 0%. The maximum percentage is 100%.
NumberOfVolumes	uint32	The total number of volumes assigned to this storage pool. This property is read-only.
Description	string	Your description of the storage pool. This property is writable. The maximum length is 256 characters. You cannot change the description for System storage pool

Create() method

Use the Create() method to define a new user storage pool. This method is the constructor for the class.

Execute Role: Administrator

Method Type: Static

Parameters

Table 84 describes the parameters that you can specify for the Create() method.

Table 84. Create() method parameters

Name	Type	Description
Name	string	Input parameter that is your label for the storage pool. The maximum length is 256 characters.
Description	string	Input parameter that is your description of the storage pool. The maximum length is 256 characters.
PartitionSize	uint32	Input parameter that is the partition size, in MB, to use when a file et allocates space. Possible values are 16, 64, and 256 MB. The default value is 16 MB.
AllocSize	uint32	Input parameter that is the allocation strategy to use for files on this storage pool: • System allocation - An escalation algorithm is used to allocate blocks to a file placed on this storage pool. Indicated by a AllocSize value of zero. This is the default value. • Fixed allocation - The file is extended by a chosen fixed size every time. The fixed allocation size is indicated by this AllocSize value, either 4 KB or 128 KB. This property is read-only. Possible values are 0, 4, or 128. The default value is 0.
AlertPercentage	uint16	Input parameter that is the percentage of the storage pool size that, when reached, causes the server to generate an alert message. The minimum value is 0%, and the maximum value is 100%.

Return values

The Create() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 9 (Name not valid; the name has invalid characters.)
- 13 (Is default; DEFAULT is a reserved name and cannot be used as a name for a user storage pool.
- 15 (Is system; SYSTEM is a reserved name and cannot be used as a name for a user storage pool.)
- 18 (Storage pool name already exists.)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)

- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

Delete() method

Use the Delete() method to delete an existing, empty, unreferenced storage pool. You cannot delete a storage pool that contains volumes or that has references in an active policy.

Execute Role: Administrator

Method Type: Dynamic

Return values

The Delete() method returns one of the following codes:

- 0 (Completed successfully)
- 5 (Storage pool is in use.)
- 8 (Integrity lost)
- 13 (Storage pool is the default storage pool.)
- 14 (The current active policy references the storage pool.)
- 15 (Storage pool is a System storage pool.)
- 21 (Storage pool not found)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

DisableDefault() method

Use the DisableDefault() method to disable the use of a default storage pool.

Execute Role: Administrator

Method Type: Dynamic

Return values

The DisableDefault() method returns one of the following codes:

- 0 (Completed successfully)
- 3 (Already disabled; the default storage pool is already disabled.)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)

- .. (Internal error)

Move() method

Use the Move() method to move or rename a storage pool by creating a new storage pool with the specified name and migrating the data and capabilities to the new name. If successful, the old storage pool will be removed.

You cannot rename a system storage pool. You cannot use SYSTEM or DEFAULT for the new name.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 85 describes the parameters that you can specify for the Move() method.

Table 85. Move() method parameters

Name	Type	Description
NewName	string	Input parameter that is your new label for the storage pool.

Return values

The Move() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 9 (Name not valid - The name has invalid characters.)
- 13 (Is default; DEFAULT is a reserved storage pool name)
- 15 (Is system; cannot rename the system storage pool or SYSTEM cannot be the new name.)
- 18 (Storage pool name already exists.)
- 21 (Storage pool not found)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

SetDefault() method

Use the SetDefault() method to change a user storage pool to the default storage pool. The STC_StoragePool class PoolType property changes from User to User Default.

Execute Role: Administrator

Method Type: Dynamic

Return values

The SetDefault() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 13 (Is already default)
- 15 (Storage pool is a system storage pool)
- 21 (Storage pool not found)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

STC_SystemMDRAid

The STC_SystemMDRAid class supports the recovery of metadata for a cluster. It provides a mechanism to extract system metadata information into a recovery file on the local disk of the system (not on the SAN). This class extends the CIM_ManagedSystemElement class.

You can extract multiple recovery files to save the state of the system metadata at various points in time. You can generate administrative commands from the recovery file, to re-create metadata using the GenerateCommandFiles() method. After you re-create the metadata, you can create the recovery file again and verify it with the original recovery file.

Properties

The STC_SystemMDRAid class has the following properties:

Table 86. STC_SystemMDRAid class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_Cluster. This property is key. The maximum length is 256 characters.
SystemName	string	The instance name of the scoping system. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_SystemMDRAid. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
Name	string	The name of the extracted metadata recovery file in the format <name>.dump. This property is key. The maximum length is 256 characters.
LocalDirectoryName	string	The metadata server local disk directory name where the recovery files and the generated command files are stored. This property is read-only. The maximum length is 256 characters.
CLIGeneratorName	string	The script used to generate command files from the metadata recovery file. This property is read-only.

Table 86. STC_SystemMDRAid class properties (continued)

Name	Type	Description
InstallDate	datetime	The date when the recovery file was created. This property is read-only.
Size	uint64	The size, in kilobytes, of the metadata recovery file. This property is read-only.

Create() method

Use the Create() method to create a new metadata recovery file. This method is the constructor for the class.

Execute Role: Administrator

Method Type: Static

Parameters

Table 87 describes the parameters you can specify for the Create() method.

Table 87. Create() method parameters

Name	Type	Description
Name	string	Input parameter that is the name of the extracted metadata recovery file in the format <name>.dump. The maximum length is 256 characters.
IsForce	boolean	Input parameter that indicates whether to overwrite an existing recovery file

Return values

The Create() method returns one of the following codes:

- 0 (Completed successfully)
- 7 (Insufficient space)
- 8 (Integrity lost)
- 9 (Invalid name)
- 18 (Recovery file name already exists)
- 22 (Not the primary administrative server)
- 30 (Transaction failed.)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

Delete() method

Use the Delete() method to delete an existing metadata recovery file.

Execute Role: Administrator

Method Type: Dynamic

Return values

The Delete() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- .. (Internal error)

GenerateCommandFiles() method

Use the GenerateCommandFiles() method to generate commands for re-creating metadata from a recovery file. Any existing command files are overwritten.

The method can generate the following set of command files:

- TankSysCLI.auto - This file contains commands to re-create storage pools, filesets, and policies. In case of disaster, this file can be run without manual intervention.
- TankSysCLI.volume - This file contains commands to re-create volumes. This file cannot be run without manual verification and editing.
- TankSysCLI.attachpoint - This file contains commands to re-create fileset attach points. This file cannot be run without manual verification, editing, and intervention.

These command files are needed only for recovery, and their generation can be postponed until needed. Also the metadata server does not have to be up and running to generate these files.

Execute Role: Administrator

Method Type: Dynamic

Return values

The GenerateCommandFiles() method returns one of the following codes:

- 0 (Completed successfully)
- 7 (Insufficient space)
- 8 (Integrity lost)
- 30 (Transaction failed)
- 22 (Not the primary administrative server)
- .. (Internal error)

STC_TankDisruptiveSetting

The STC_TankDisruptiveSetting class contains the settings for server-specific configuration parameters that need a metadata server restart for an update to take effect. These parameters are read-only and can only be specified on the command line when the server is started. This class extends the STC_Setting class.

Properties

The STC_TankDisruptiveSetting class has the following properties:

Table 88. STC_TankDisruptiveSetting class properties

Name	Type	Description
ServerName	string	The unique server name that can be set only at installation. This property is read-only. The maximum length is 32 characters.
ProtocolType	uint32	The client-server and server-server communication protocol type. • 0: UDP • 1: TCP Default is 0.
ClientNetwork Protocol	uint32	The client-server communication protocol type. This property is read-only. Possible values are: • 0: UDP • 1: TCP Default is 1.
ServerNetwork Protocol	uint32	The server-server communication protocol type. This property is read-only. Possible values are: • 0: UDP • 1: TCP Default is 0.
NumDeleteThr eads	uint32	The number of threads for garbage collection of deleted files. This property is read-only. The minimum value is 1 and the maximum value is 4.
PrimaryIP	string	The IP address of the Ethernet interface for the server. This property is read-only.
ClusterPort	uint32	The cluster port used by internal group services infrastructure communication. This port must be free on the interface when the server is started. This property is read-only. The minimum value is 1 024 and the maximum value is 65 535. The default value is 1 737.
HeartbeatPort	uint32	The heartbeat port used by internal group services infrastructure communication on which to receive heartbeats. This port must be free on the interface when the server is started. This property is read-only. Minimum is 1 024 and maximum is 65 535. Default is 1 738.
STPPort	uint32	The SAN File System protocol port used for communication with file system clients. This port must be free on the interface when the server is started. This property is read-only. The minimum value is 1 024 and the maximum value is 65 535. The default value is 1 700.
AdminPort	uint32	The port to receive administrative requests. This port must be free on the interface when the server is started. This property is read-only. The minimum value is 1 024 and the maximum value is 65 535. The default value is 1 800.
NoLogReserve	boolean	An indicator of whether the metadata server should reserve disk space for logs. The property is writable. The default value is True.

STC_TankEvents

The STC_TankEvents class represents the possible events that a metadata server can generate. This class extends the CIM_LogicalElement class.

A master metadata server can generate any of these events and a subordinate metadata server can generate only a subset. An instance of this class is not the list of events that occurred in a server, just the possible events that could occur.

SNMP traps are either generic or specific traps as indicated by the SNMPTrap property. The meaning of a generic trap can be interpreted by the message content. Thus, only generic traps contain the message actually logged in the event log.

A specific trap is specific to a particular event. A specific trap does not contain the message content. Rather, the name of the trap itself indicates what specific event has occurred. For example, the tankClusterStateChangeTrap trap is generated whenever the primary administrative server changes the cluster state. This trap contains the OldState and the CurrentState values of the cluster. Each specific trap also contains context (varbinds) that further identifies what happened. Some other specific traps are:

- tankLogRotateTrap - The metadata server log has been rotated.
- tankStoragePoolSpaceTrap - The storage pool usage has exceeded its alert percentage with a new allocation.
- tankContainerQuotaTrap - Fileset hard or soft quota violation.

The SNMP Trap MIB defines the specific traps.

Properties

The STC_TankEvents class has the following properties:

Table 89. STC_TankEvents class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system. This property is key. The maximum length is 256 characters.
SystemName	string	The name of the scoping system. This property is key. The maximum length is 256 characters.
ServiceCreationClassName	string	The class name of the scoping service. This property is key. The maximum length is 256 characters.
ServiceName	string	The name of the scoping service. This property is key. The maximum length is 256 characters.
MessageID	string	The ID associated with the message that will be logged in the metadata server log when this event occurs. This property is key.
Severity	uint8	The severity level of the event. This property is read-only. Possible values are: • 0: Information • 1: Warning • 2: Error • 3: Severe

Table 89. STC_TankEvents class properties (continued)

Name	Type	Description
Message	string	The message format the metadata server will use to log a message when this event occurs in the server. If there are any parameter format specifications in this string, they are replaced with actual values when this message is logged. This property is read-only.
SNMPTrap	string	The name of the SNMP trap generated by this event. All instances with this property value set to tankGenericTrap are generic traps. The rest of the instances are specific traps. This property is read-only.
IsSNMPTrapEnabled	boolean	An indicator of whether the SNMPEvents configuration parameter filter allows the generation of the SNMP trap if this event occurs. This property is read-only.

Test() method

Use the Test() method to generate a test event. You can check that an SNMP manager can receive a trap.

This event causes a trap with severity information. Make sure that the following conditions are met:

- SNMPEvents configuration parameter is set to allow event with severity information.
- SNMPManagers configuration parameter is also set properly.
- Specified SNMP managers are configured properly and active to receive traps. The SNMPManagers property in the "STC_MasterDynamicSetting" on page 109 class lists the SNMP managers' IP addresses.

Execute Role: Administrator

Method Type: Static

Return values

The Test() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- .. (Internal error)

STC_TankMetrics

The STC_TankMetrics class represents the metrics for each metadata server. Only one instance of this class should exist for each server running including the master metadata server. This class extends the CIM_ServiceStatisticalInformation class.

The metrics include the current totals for the following types of buffers:

- Clean - Buffers that contain data but are available for reuse.
- Dirty - Buffers that contain data that is awaiting I/O to disk.
- Free - Buffers that are available because they are currently not in use.

Properties

The STC_TankMetrics class has the following properties:

Table 90. STC_TankMetrics class properties

Name	Type	Description
TotalUserMetaActivity	uint64	The total number of transactions relating to file system metadata activity including fileset attach points, creating directories, and extending files. This property is read-only.
TotalUserMetaUpdateActivity	uint64	The total number of transactions relating to file system updates for system objects. This property is read-only.
SessionLocks	uint64	The current number of session locks held in the Lock Manager for this server. Session lock holds a reference in each file that it manages. You must acquire a session lock to perform any actions with the file, such as a stat, lstat, or opendir operation. This property is read-only.
DataLocks	uint32	The current number of data locks held in the Lock Manager. This property is read-only.
ByteRangeLocks	uint32	The current number of byte range locks held in the Lock Manager. This property is read-only.
TotalBuffers	uint32	The current number of total buffers for user metadata activity. This property is read-only.
CleanBuffers	uint32	The current number of clean buffers for user metadata activity. Clean buffers contain data but the buffers are available for reuse. This property is read-only.
DirtyBuffers	uint32	The current number of dirty buffers for user metadata activity. Dirty buffers contain data awaiting I/O to disk. This property is read-only.
FreeBuffers	uint32	The current number of free buffers for user metadata activity. Free buffers are available because they are currently not in use. This property is read-only.

STC_TankSAP

The STC_TankSAP class represents a metadata server service access point. It extends the CIM_ServiceAccessPoint class.

Properties

The STC_TankSAP class has the following properties:

Table 91. STC_TankSAP class properties

Name	Type	Description
TypeOfAddress	uint16	An enumeration that defines how to format the address and mask of the address range that defines this IP subnet. Whenever possible, IPv4-compatible addresses should be used instead of IPv6 addresses (see RFC 2373, section 2.5.4). To have a consistent format for IPv4 addresses in a mixed IPv4 and IPv6 environment, all IPv4 addresses and both IPv4-compatible IPv6 addresses and IPv4-mapped IPv6 addresses, per RFC 2373, section 2.5.4, should be formatted in standard IPv4 format. However, the 2.2 version of the Network Common Model will not explicitly support mixed IPv4 and IPv6 environments. This support will be added in a future release. This property is read-only. Possible values are: 0: Unknown 1: IPv4 2: IPv6 The default value is 1.
Ip	string	The IP address of the Ethernet network interface of a metadata server. The Group Services and the SAN File System protocol are bound to this IP at boot time. The HeartBeat protocol and the Admin Service are also bound to this port for service. This property is read-only.
RSAIp	string	The IP address for the Remote Supervisory Adapter (RSA) card on the engine. This property is read-only. Default is 0.0.0.0.
ClusterPort	uint32	The cluster port used by internal group services infrastructure communication. This port must be free on both the interfaces when the server is started. This property is read-only. The default value is 1 737. The minimum value is 1 024. The maximum value is 65 535.
HeartbeatPort	uint32	The heartbeat port used by internal infrastructure communication to receive heartbeats. This port must be free on both the interfaces when the engine is started. This property is read-only. The default value is 1 738. The minimum value is 1 024. The maximum value is 65 535.
STPPort	uint32	The SAN File System protocol port used for communication with file system clients. This port must be free on both the interfaces when the engine is started. This property is read-only. The default value is 1 700. The minimum value is 1 024. The maximum value is 65 535.
AdminPort	uint32	The port to receive administrative requests. This port must be free on both the interfaces when the engine is started. This property is read-only. The minimum value is 1 024. The maximum value is 65 535.
IsLocal	boolean	An indicator of whether the local server is the master metadata server in the cluster.

STC_TankService

The STC_TankService class represents a metadata server and provides server services. It extends the CIM_Service class.

Properties

The STC_TankService class has the following properties:

Table 92. STC_TankService class properties

Name	Type	Description
CurrentState	uint32	The state of the metadata server. This property is read-only. Possible values are: 0: Down 1: Online 2: Partly Quiescent - Only server I/O operations are suspended. 3: Fully Quiescent - All background I/O, client, and server operations are suspended. 4: Administrative Quiescent - No longer servicing clients. 5: Initializing for the first time 6: FailedInit - Encountered an error during startup or group formation. 7: UnCommissioned - Not commissioned into a cluster. 8: Joining a cluster. 9: Unknown
PendingState	uint32	The current state of the server transitions to this state if the current state is different from this pending state. This property is read-only. Possible values are the same as the CurrentState property.
LastBootUpTime	datetime	The time when the server was last started. This property is read-only.
LocalDateTime	datetime	The local date and time of day according to the server. This property is read-only.
LastCurrentStateChangeTime	datetime	The time since the server changed its current state. This property is read-only.
LastPendingStateChangeTime	datetime	The time since the server has a state change pending. This property is read-only.
CurrentVersion	string	The current software release version. This is the version of the latest upgrade. This will be different from the previous (committed) version if there was an upgrade but the commit was not yet activated. This property is read-only.
IsMaster	boolean	An indicator of whether this is the master metadata server. This property is read-only.
NumberOfContainers	uint32	The number of filesets served from this server. This property is read-only.

BecomeMaster() method

Use the BecomeMaster() method to make this server the master metadata server, if the master metadata server is irrecoverably lost. A master can be lost because of hardware failures, software failures, or partitioned networks.

Note:

1. This method is not available in SAN File System version 2.1. It is available only during a rolling upgrade from SAN File System version 1.1 to version 2.1.
2. You must ensure that the previous master is down to prevent rogue server issues. Turn the power off to the engine hosting the master metadata server that is down.
3. You cannot change the master when the cluster is down.
4. When a master is lost, the subordinate servers are no longer in operational states.
5. You must manually log in to the metadata server and edit the STC_TankService class property file to set the IsMaster property to True.

Execute Role: Administrator

Method Type: Dynamic

Return values

The BecomeMaster() method returns one of the following codes:

- 0 (Completed successfully. This server has become the master metadata server.)
- 8 (Integrity lost)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 58 (The server is already the master metadata server.)
- 59 (The server is a subordinate, but it is not in the right state to become the master metadata server.)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 67 (Cannot become the primary administrative server)
- .. (Internal error)

StartService() method

Use the StartService() method to start the metadata server on this engine.

Execute Role: Administrator

Method Type: Dynamic

Return values

The StartService() method returns one of the following codes:

- 0 (Completed successfully; the metadata server started.)
- 1 (Not supported)
- 4 (Command failed)
- 5 (In use)
- 8 (Integrity lost)
- 21 (Not found)
- 24 (Server timed out)
- 30 (Transaction failed)

- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 66 (Metadata server restart service state cannot continue)
- 78 (Aborted)
- 91 (Cannot contact remote CIM agent)
- .. (Internal error)

StopService() method

Use the StopService() method to stop the metadata server on this engine.

Execute Role: Administrator

Method Type: Dynamic

Return values

The StopService() method returns one of the following codes:
- 0 (Completed successfully)
- 1 (Not supported)
- 4 (Command failed)
- 8 (Integrity lost)
- 21 (Not found)
- 24 (Server timed out)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 66 (Metadata server restart service state cannot continue)
- 91 (Cannot contact remote CIM agent)
- .. (Internal error)

STC_TankTransientSetting

The STC_TankTransientSetting class contains server-specific configuration parameters that are effective only until the next restart. This class extends the STC_Setting class.

STC_TankWatchdog

The STC_TankWatchdog class represents the metadata server restart service operations. It extends the CIM_LogicalElement class.

There is an instance of this class for each metadata server in the cluster. This administrative service keeps track of the vitality of a metadata server. If enabled, the service restarts the server when it detects positively that the server is down.

Properties

The STC_TankWatchdog class has the following properties:

Table 93. STC_TankWatchdog class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_Cluster. This property is key. The maximum length is 256 characters.
SystemName	string	The name of the STC_Cluster instance that is the scoping system. This property is key. The maximum length is 256 characters.
ServiceCreationClassName	string	The class name of the scoping service: STC_TankService. This property is key. The maximum length is 256 characters.
ServiceName	string	The name of the STC_TankService instance that is the scoping service. This property is key. The maximum length is 256 characters.
CreationClassName	string	The name of the class or the subclass used in the creation of an instance: STC_TankWatchdog. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified. This property is key. The maximum length is 256 characters.
Name	string	An address or other identifying information to uniquely name the metadata server restart service. This property is key. The maximum length is 256 characters.
State	uint32	The state of the metadata server restart service. This property is read-only. Possible values are: 0: Off - The metadata server restart service is manually turned off. 1: On - The metadata server restart service is manually turned on. 2: Standby - The metadata server restart service is in a passive standby mode because the server it is probing has been manually shut down. The watchdog automatically turns on when this server is restarted. 3: Aborted - The metadata server restart service reached the retry limit for detecting server liveness and the metadata server restart service was turned off. An administrator must manually turn on the watchdog again to continue. Unknown - The metadata server restart service is in an indeterminate state because the metadata server restart service server could not be reached. The state of the metadata server restart service persists with a server restart. Default is 0.

Table 93. STC_TankWatchdog class properties (continued)

Name	Type	Description
ProbeState	uint32	The server status found in the current probe cycle. This property is read-only. Possible values are: 0: Not Probed - Metadata server restart service has not started the probe because it is off or in a standby or an aborted state 1: Probing - Metadata server restart service has started a probe. 2: Server Live - Metadata server restart service detected that the server is live. There is no need to restart the server. 3: Server Absent - Metadata server restart service positively detected that the server is absent. The metadata server restart service attempts to restart the server. Unknown - The liveness test failed and the absence test failed. The metadata server restart service does not attempt to restart the server. The default value is 0.
ProbeInterval	uint32	The interval, in seconds, at which the metadata server restart service will periodically start a probe. This property is read-only. The minimum value is 10 seconds and the maximum value is 60. The default value is 10.
LiveTestTimeoutInterval	boolean	The maximum time, in seconds, to wait for the server to respond to a liveness test request before deciding that the server is not live. This property is read-only. The minimum value is 1 second and the maximum value is 10 seconds. The default value is 2.
RetryLimit	uint32	The number of times to try detecting liveness of the server if the server is declared not live. When this retry limit is reached, the metadata server restart service is turned off. This property is the number of tries, not the number of retries. For example, if this value is 3, the probe is sent three times, the original time plus two retries. This property is read-only. The minimum value is 1 and the maximum value is 10. The default value is 3.
The following properties represent statistics initialized when watchdog becomes active.		
StartTimeStamp	datetime	The date and time when the metadata server restart service was started. This property is read-only.
LastProbeTimeStamp	datetime	The date and time when the last probe was started. This property is read-only.
TotalProbes	uint64	The counter of the total number of probes done so far. This property is read-only.
LiveTestTimeouts	uint64	The counter of the total number of times the liveness test has taken longer than the test timeout interval and caused a timeout error to occur. This property is read-only.
The following properties represent statistics for retries.		
TotalRetries	uint64	The counter of the total number of retries done so far. This property is read-only.
CurrentRetries	uint32	The number of retries in the current probe cycle. This property is read-only. The value of this property ranges from 0 to one less than the limit as set by the RetryLimit property.
RetriesLWM	uint32	The lowest number of retries reached so far. This property is read-only.

Table 93. STC_TankWatchdog class properties (continued)

Name	Type	Description
RetriesHWM	uint32	The highest number of retries reached so far. This property is read-only.
The following properties represent statistics for liveness tests.		
LastLiveTestTime	uint32	The time taken by the last liveness test. This property is read-only.
LiveTestTimeLWM	uint32	The low watermark for time, in milliseconds, taken by the liveness test. This property is read-only.
LiveTestTimeHWM	uint32	The high watermark for time, in milliseconds, taken by the liveness test. This property is read-only.
The following properties represent statistics absence tests.		
TotalAbsenceTests	uint64	The counter of the total number of time absence test was started. This property is read-only.
LastAbsenceTestTime	uint32	The time, in milliseconds, taken by the last absence test. This property is read-only.
AbsenceTestTimeLWM	uint32	The low watermark time, in milliseconds, taken by the last absence test. This property is read-only.
AbsenceTestTimeHWM	uint32	The high watermark for time, in milliseconds, taken by the last absence test. This property is read-only.

Disable() method

Use the Disable method to disable the metadata server restart service.

Execute Role: Administrator

Method Type: Dynamic

Return values

The Disable() method returns one of the following codes:

- 0 (Completed successfully)
- 61 (Cannot connect to metadata server)
- 64 (Metadata-server restart service is already disabled)
- 66 (Cannot continue metadata-server restart service)
- .. (Internal error)

Enable() method

Use the Enable() method to enable the metadata server restart service.

Execute Role: Administrator

Method Type: Dynamic

Return values

The Enable() method returns one of the following codes:

- 0 (Completed successfully)
- 61 (Cannot connect to metadata server)

- 63 (Metadata server restart service is already enabled)
- 66 (Cannot continue metadata server restart service)
- .. (Internal error)

STC_Volume

The STC_Volume class represents a volume. The list of instances and the methods defined in this class are available only on the master metadata server. This class extends the CIM_ManagedSystemElement class.

A volume is a logical unit number (LUN) labeled by SAN File System for its use. You can use the"STC_AvailableLUNs" on page 89 class to see all the LUNs available on a host engine.

Properties

The STC_Volume class has the following properties:

Table 94. STC_Volume class properties

Name	Type	Description
StoragePoolName	string	The storage pool to which this volume belongs. This property is read-only.
Caption	string	A one-line description of the object. This property is read-only. The maximum length is 64 characters.
Name	string	Your label for the volume. This property is key. Its alias is VolumeName. The maximum length is 256 characters.
OSDeviceName	string	The file path to the storage device. This property is read-only. The maximum length is 256 characters.
State	uint32	The state of the volume. This property is read-only. Possible values are: 0: Normal - Volume is available for reading, writing and allocation. 1: Suspend Allocations - Allocation of new partitions to filesets is suspended. A client can still read from and write to the volume and a fileset can still allocate a new file on this volume if sufficient space exists. 2: Volume being Deleted - Volume is processing a deletion request.
Size	uint64	The size, in megabytes, of the volume. This property is read-only.
SizeAllocated	uint64	The size, in megabytes, of the volume allocated to filesets. This property is read-only.
SizeAllocatedPercentage	uint16	The percentage of the size allocated in the volume. This property is read-only. The minimum value is 0% and the maximum value is 100%.
Description	string	Your description of the volume. This property is writable. The maximum length is 256 characters.

Create() method

Use the Create() method to create a volume using a device that is accessible by a metadata server or to attach a volume to a storage pool. This method is the constructor for the class.

Execute Role: Administrator

Method Type: Static

Parameters

Table 95 describes the parameters that you can specify for the Create() method.

Table 95. Create() method parameters

Name	Type	Description
OSDeviceName	string	Input parameter that is the file path to the storage device. The maximum length is 256 characters.
StoragePoolName	string	Input parameter that is your label for the storage pool to which you are attaching the volume. The maximum length is 256 characters.
VolumeName	string	Input parameter that is your label for the volume. The maximum length is 256 characters.
Description	string	Input parameter that is your description of the volume. The maximum length is 256 characters.
IsForce	boolean	Input parameter that indicates whether a volume will be deleted even if it has files on it.
IsSuspendAllocations	boolean	Input parameter that indicates whether the volume state is set to Suspend Allocations.

Return values

The Create() method returns one of the following codes:

- 0 (Completed successfully)
- 3 (Already defined; the storage device has a label and IsForce parameter is set to True but the device is already defined as another volume.)
- 5 (In use; the storage device has a label and IsForce parameter is set to False.)
- 8 (Integrity lost)
- 9 (Volume name is not valid)
- 10 (Parameter not valid)
- 12 (Storage device I/O failed)
- 18 (Volume name already exists.)
- 21 (Storage device not found)
- 22 (Not the primary administrative server)
- 28 (Storage pool not found)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 52 (Disk not viable for one of the following reasons:
  - The volume does not hold at least one partition
  - The specified local storage device is not viable as a global disk
  - Hashing using world-wide name (WWN) conflicts with an existing hash
  - Sector size is less than 512 or greater than 4096
  - If a volume is being added to the system storage pool, the sector sizes of all volumes are not the same.
- 56 (Storage device access denied)

- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

If errors such as I/O failed, invalid size, or internal error occur, you can decommission the disk by deleting the volume.

CreateUsingLunId() method

Use the CreateUsingLunId() method to create a volume using a specified LUN. The volume creation is performed by a client, if specified, or by the metadata server. This method is the constructor for the class.

Execute Role: Administrator

Method Type: Static

Parameters

Table 96 describes the parameters that you can specify for the CreateUsingLunId() method.

Table 96. CreateUsingLunId() method parameters

Name	Type	Description
LunID	string	Input parameter that is the identifier of the LUN.
ClientName	string	Input parameter that is the name of a client that can access the volume. The maximum length is 256 characters.
StoragePoolName	string	Input parameter that is your label for the storage pool to which you are attaching the volume. The maximum length is 256 characters.
VolumeName	string	Input parameter that is your label for the volume. The maximum length is 256 characters.
Description	string	Input parameter that is your description of the volume. The maximum length is 256 characters.
IsForce	boolean	Input parameter that indicates whether a volume will be deleted even if it has files on it.
IsSuspendAllocations	boolean	Input parameter that indicates whether the volume state is set to Suspend Allocations.

Return values

The CreateUsingLunId() method returns one of the following codes:

- 0 (Completed successfully)
- 3 (Already defined; the storage device has a label and IsForce parameter is set to True but the device is already defined as another volume.)
- 5 (In use; the storage device has a label and IsForce parameter is set to False.)
- 8 (Integrity lost)
- 9 (Volume name is not valid)
- 10 (Parameter not valid)
- 12 (Storage device I/O failed)
- 18 (Volume name already exists.)

- 21 (Storage device not found)
- 22 (Not the primary administrative server)
- 28 (Storage pool not found)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 48 (Disk not found)
- 52 (Disk not viable for one of the following reasons:
  - The volume does not hold at least one partition
  - The specified local storage device is not viable as a global disk
  - Hashing using world-wide name (WWN) conflicts with an existing hash
  - Sector size is less than 512 or greater than 4096
  - If a volume is being added to the system storage pool, the sector sizes of all volumes are not the same.
- 56 (Storage device access denied)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 79 (System volume; a client name was specified to a volume in the system storage pool.)
- 81 (Client not found; the specified client name does not match any identified clients.)
- 82 (Client not reachable)
- 83 (Client I/O failed)
- 84 (Client unsupported operation)
- 88 (Rogue client)
- 89 (Invalid LUN ID)
- 90 (Use client; the metadata server cannot access the specified LUN.)
- .. (Internal error)

If errors such as I/O failed, invalid size, or internal error occur, you can decommission the disk by deleting the volume.

Delete() method

Use the Delete() method to delete an existing volume that is accessible from a metadata server.

Use the Delete() method to delete an existing volume. If the IsForce parameter is True, this method deletes all the files that partly or fully exist on the volume before it deletes the volume. If the IsForce parameter is False, this method drains the volume first by moving the file data that resides on the volume to other volumes in the same storage pool. If the volume drain fails, the volume enters the Suspend Allocations state requiring manual administrative action.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 97 on page 168 describes the parameters that you can specify for the Delete() method.

Table 97. Delete() method parameters

Name	Type	Description
IsForce	boolean	Input parameter that indicates whether a volume will be deleted even if it has files on it.

Return values

The Delete() method returns one of the following codes:

- 0 (Completed successfully)
- 2 (Access failed)
- 5 (In use; volume drain failed and volume is not empty.)
- 8 (Integrity lost)
- 12 (I/O failed)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 33 (Volume not found)
- 44 (Incompatible operation)
- 56 (Access denied)
- 57 (No space; no space in other volumes of the storage pool for volume drain.)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 65 (Server state offline; force option removes files from filesets. The serving server must be online. Without force, volume is drained; you need to revoke locks; the serving fileset must be up.)
- .. (Internal error)

DeleteUsingClient() method

Use the DeleteUsingClient() method to delete an existing volume. The volume deletion is performed by a client, if specified, or by the metadata server.

Use the DeleteUsingClient() method to delete an existing volume. If the IsForce parameter is True, this method deletes all the files that partly or fully exist on the volume before it deletes the volume. If the IsForce parameter is False, this method drains the volume first by moving the file data that resides on the volume to other volumes in the same storage pool. If the volume drain fails, the volume enters the Suspend Allocations state requiring manual administrative action.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 98 describes the parameters that you can specify for the DeleteUsingClient() method.

Table 98. DeleteUsingClient() method parameters

Name	Type	Description
ClientName	string	Input parameter that is the name of a client that can access the volume. The maximum length is 256 characters.

Table 98. DeleteUsingClient() method parameters (continued)

Name	Type	Description
IsForce	boolean	Input parameter that indicates whether a volume will be deleted even if it has files on it.

Return values

The DeleteUsingClient() method returns one of the following codes:

- 0 (Completed successfully)
- 2 (Access failed)
- 5 (In use; volume drain failed and volume is not empty.)
- 8 (Integrity lost)
- 12 (I/O failed)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 33 (Volume not found)
- 44 (Incompatible operation)
- 48 (Disk not found)
- 56 (Access denied)
- 57 (No space; no space in other volumes of the storage pool for volume drain.)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 65 (Server state offline; force option removes files from filesets. The serving server must be online. Without force, volume is drained; you need to revoke locks; the serving fileset must be up.)
- 79 (System volume; a client name was specified to a volume in the system storage pool.)
- 81 (Client not found; the specified client name does not match any identified clients.)
- 82 (Client not reachable)
- 83 (Client I/O failed)
- 84 (Client unsupported operation)
- 88 (Rogue client)
- 90 (Use client; the metadata server cannot access the specified LUN.)
- .. (Internal error)

GetNextFOV() method

Use the GetNextFOV() method to get the next file on volume (FOV) entry, given the FOV iterator handle FOVHandle.

Use the GetNextFOV() method to get the next file on volume (FOV) entry, by providing the FOV iterator in the FOVHandle input parameter. The file entry is made available in the FOVEntry output parameter. The iterator to use for the next call is returned in the FOVHandle parameter.

Note: This method can be a long-running process depending on the number of files and size of storage.

Execute Role: Backup

Method Type: Dynamic

Parameters

Table 99 describes the parameters that you can specify for the GetNextFOV() method

Table 99. GetNextFOV() method parameters

Name	Type	Description
FOVHandle	string	The input/output parameter that is the FOV iteration identifier.
FOVEntry	string	The output parameter that is the file entry.

Return values

The GetNextFOV() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 37 (End of Iteration; end of file reached.)
- 38 (Invalid IterationIdentifier)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

Move() method

Use the Move() method to move or rename a volume. This method creates a new volume with the specified new name and migrates the data and capabilities to the new volume. If successful, this method deletes the old volume.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 100 describes the parameters that you can specify for the Move() method:

Table 100. Move() method parameters

Name	Type	Description
NewName	string	Input parameter that is your new label for the volume.

Return values

The Move() method returns one of the following codes:

- 0 (Completed successfully)
- 8 (Integrity lost)

- 9 (Name not valid; the new name has invalid characters.)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

ResetFOV() method

Use the ResetFOV() method to reset a Files-On-Volume (FOV) iterator. This method creates and returns an iterator that can be used to locate each file entry that resides on this volume instance.

You pass in the iterator when you invoke the "GetNextFOV() method" on page 169, which returns one successive file entry per call. Only one iterator can be active at any given instance.

Note: This method can be a long-running process depending on the number of files and size of storage.

Execute Role: Backup

Method Type: Dynamic

Parameters

Table 101 describes the parameters that you can specify for the ResetFOV() method:

Table 101. ResetFOV() method parameters

Name	Type	Description
FOVHandle	string	The output parameter that is the FOV iteration identifier.

Return values

The ResetFOV() method returns one of the following codes:

- 0 (Completed successfully)
- 2 (Access failed)
- 7 (Insufficient space; not enough temporary space on the local disk.)
- 8 (Integrity lost)
- 21 (Not found; no files found on this volume.)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 44 (Incompatible operation)
- 56 (Access denied)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 65 (Server state offline; the volume has files of a fileset with its server offline.)
- .. (Internal error)

Resize() method

Use the Resize() method to notify a metadata server that the capacity of the LUN has increased. This notification enables the metadata server to expand the size of the volume.

Execute Role: Administrator

Method Type: Dynamic

Parameters

Table 102 describes the parameters that you can specify for the Resize() method.

Table 102. Resize() method parameters

Name	Type	Description
ClientName	string	The input parameter that is the name of a client that can access the volume. This parameter is required only if the volume is not accessible by a metadata server. When this parameter is specified, the method uses the client regardless of whether a metadata server can access the LUN.

Return values

The Resize() method returns one of the following codes:

- 0 (Completed successfully)
- 11 (Invalid size)
- 22 (Not the primary administrative server)
- 44 (Incompatible operation)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- 69 (Up-to-date)
- 79 (System volume)
- 80 (Device not found)
- 81 (Client not found)
- 82 (Client not reachable)
- 83 (Client I/O failed)
- 84 (Client unsupported operation; the client is running an older version of SAN File System that does not support this method.)
- 85 (File system check resized; the file system check has already noticed the resize and updated the volume.)
- .. (Internal error)

ResumeAllocation() method

Use the ResumeAllocation() method to resume suspended partition allocations on a volume.

Use the ResumeAllocation() method to resume suspended partition allocations on a volume.

Execute Role: Administrator

Method Type: Dynamic

Return values

The ResumeAllocation() method returns one of the following codes:

- 0 (Completed successfully)
- 2 (Access failed)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 33 (Volume not found)
- 35 (Allocations were not suspended.)
- 44 (Incompatible operation)
- 56 (Access denied)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

SuspendAllocation() method

Use the SuspendAllocation() method to suspend partition allocations on a volume.

Use the SuspendAllocation() method to suspend partition allocations on a volume. A metadata server cannot allocate new data on the volume.

Execute Role: Administrator

Method Type: Dynamic

Return values

The SuspendAllocation() method returns one of the following codes:

- 0 (Completed successfully)
- 2 (Access failed)
- 8 (Integrity lost)
- 22 (Not the primary administrative server)
- 30 (Transaction failed)
- 33 (Volume not found)
- 34 (Allocations already suspended.)
- 44 (Incompatible operation)
- 56 (Access denied)
- 61 (Cannot connect to server)
- 62 (Too many connections)
- .. (Internal error)

STC_WatchdogDynamicSetting

The STC_WatchdogDynamicSetting class contains the settings for metadata-server restart-service configuration parameters that you can dynamically update, without a cluster restart. These parameters persist across cluster restarts. If you have Administrator privileges, you can change the writable properties in this class using the SetProperty() intrinsic method. This class extends the STC_Setting class.

Properties

The STC_WatchdogDynamicSetting class has the following properties:

Table 103. STC_WatchdogDynamicSetting class properties

Name	Type	Description
SystemCreationClassName	string	The class name of the scoping system: STC_Cluster. This property is key. The maximum length is 256 characters.
ServiceCreationClassName	string	The class name of the scoping service: STC_WatchdogDynamicSetting. This property is key. The maximum length is 256 characters.
ServiceName	string	The name of the STC_WatchdogDynamicSetting instance that is the scoping service. This property is key. The maximum length is 256 characters.
ProbeInterval	uint32	The interval, in seconds, at which the metadata-server restart service will periodically start a probe. This property is writable. The minimum value is 10 seconds and the maximum value is 60 seconds. The default value is 10 seconds.
LiveTestTimeoutInterval	boolean	The maximum time, in seconds, to wait for the server to respond to an activity test request before deciding that the server is inactive. This property is writable. The minimum value is 1 second and the maximum value is 10 seconds. The default value is 2 seconds.
RetryLimit	uint32	The number of times to try detecting server activity if the server is declared inactive. When this number is reached, the metadata-server restart service is turned off. This property is the number of tries, not the number of retries; for example, if this value is 3, the probe is sent three times. This property is writable. The minimum value is 1 and the maximum value is 10. The default value is 3.
MaximumTransitionInterval	uint32	The maximum time, in seconds, that the provider should wait for the server to make the transition to a specific set of server states. For example, the provider needs to wait for the master server to become operational before starting other subordinate servers. This property is writable. The minimum value is 1 and the maximum value is 3600. The default value is 600.

Appendix A. Accessibility

This topic provides information about the accessibility features of SAN File System and its accompanying documentation.

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully.

Features

These are the major accessibility features in SAN File System:

- You can use screen-reader software and a digital speech synthesizer to hear what is displayed on the screen.

  Note: The SAN File System Information Center and its related publications are accessibility-enabled for the IBM Home Page Reader.
- You can operate all features using the keyboard instead of the mouse.

Navigating by keyboard

You can use keys or key combinations to perform operations and initiate many menu actions that can also be done with a mouse. You can navigate the SAN File System console and help system from the keyboard by using the following key combinations:

- To traverse to the next link, button or topic, press Tab inside a frame (page).
- To expand or collapse a tree node, press Right Arrow or Left Arrow, respectively.
- To move to the next topic node, press Down Arrow or Tab.
- To move to the previous topic node, press Up Arrow or Shift+Tab.
- To scroll all the way up or down, press Home or End, respectively.
- To go back, press Alt+Left Arrow
- To go forward, press Alt+Right Arrow.
- To go to the next frame, press Ctrl+Tab. There are quite a number of frames in the help system.
- To move to the previous frame, press Shift+Ctrl+Tab.
- To print the current page or active frame, press Ctrl+P.

Appendix B. SNMP trap MIB

The SAN File System SNMP MIB defines the structure and content of events.

It defines the following traps:

- sanfsGenericTrap
- sanfsClusterStateChangeTrap
- sanfsServerStateChangeTrap
- sanfsStoragePoolSpaceTrap
- sanfsFilesetQuotaTrap
- sanfsLogRotateTrap

As defined in the MIB, the traps capture information about SAN File System resources at the time of an event. Refer to the actual MIB for details.

Appendix C. Notices

Availability Notice

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
MW9A/050
5600 Cottle Road
San Jose, CA 95193
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Trademarks

The following terms are trademarks of International Business Machines Corporation or Tivoli Systems Inc. in the United States or other countries or both:

AIX
Enterprise Storage Server
IBM
IBM logo
FlashCopy
StorageTank
TotalStorage
WebSphere

Microsoft, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Appendix D. Glossary

This glossary includes terms and definitions from:

- *The American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions are identified by the symbol (A) after the definition.
- The *ANSI/EIA Standard - 440A: Fiber Optic Terminology*, copyright 1989 by the Electronics Industries Association (EIA). Copies can be purchased from the Electronics Industries Association, 2001 Pennsylvania Avenue N.W., Washington, D.C. 20006. Definitions are identified by the symbol (E) after the definition.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.
- The *Storage Networking Dictionary*, available online at the Storage Networking Industry Association (SNIA) Web site:

  www.snia.org/education/dictionary/
- The Distributed Management Task Force (www.dmtf.org), copyright 2003 by the Distributed Management Task Force, Inc., 225 SE Main Street Portland, OR 97214. Definitions derived from this book have the symbol (D) after the definition.

This glossary uses the following cross-reference forms:

See This refers the reader to one of two kinds of related information:

- A term that is the expanded form of an abbreviation or acronym. This expanded form of the term contains the full definition.
- A synonym or more preferred term

See also
This refers the reader to one or more related terms.

ACLI See *administrative command-line interface (ACLI)*.

administrative command-line interface (ACLI)
A command-line interface used to administer all aspects of the SAN File System. The ACLI runs on all engines that host metadata servers and the administrative server.

administrative log
A log that maintains a history of messages created by the administrative server.

administrative server
For SAN File System, a set of servlets running within a customized instance of WebSphere® Application Server that handles all SAN File System administrative requests from the SAN File System console. See also *SAN File System console*.

alert A message or other indication that identifies a problem or an impending problem.

audit log
A log that maintains the history of all commands issued by any administrator for all metadata servers in the cluster.

CIM See *Common Information Model*.

CIM client application
A storage management program that initiates CIM requests to the administrative agent for the device.

CIM namespace
The scope within which a CIM schema applies.

CIM object manager (CIMOM)
The common conceptual framework for data management that receives, validates, and authenticates the CIM requests from the client application and then directs the requests to the appropriate component or device provider.

CIMOM
See *CIM object manager*.

client For SAN File System, a client is a system that can access the SAN File System. These clients act as servers to a broader clientele, providing Network File System or Common Internet File System access to the global namespace or hosting applications (such as database servers or Web-hosting services that use multiple servers).

class The definition of an object within a specific hierarchy. An object class can have properties and methods and serve as the target of an association.

CLI See *Administrative command-line interface*.

client state manager (CSM)
A component of the client kernel that provides protocol support for the client.

cluster
A group of engines that is managed as a set and presents a single point of control for configuration and service activity.

cluster log
A log that maintains a history of messages created by all metadata servers in the cluster.

cluster state
A status condition of the cluster. Cluster states can be inactive (Not running or Forming), active (Online, Offline, Partly quiescent, or Fully quiescent) or unknown. See also *Forming, Fully quiescent, Not running, Offline, Online,* and *Partly quiescent*.

Common Information Model (CIM)
A set of standards from the Distributed Management Task Force Inc. (DMTF). CIM provides a conceptual framework for storage management and an open approach to the design and implementation of storage systems, applications, databases, networks, and devices.

coordinated universal time (UTC)
The time scale, based on the System International (SI) second, as defined

and recommended by the Comitb Consultatif International de la Radio (CCIR) and maintained (using an atomic clock) by the Bureau International des Poids et Mesures (BIPM).

CSM See *client state manager*.

default user storage pool
A storage pool that stores file data that SAN File System has not assigned (using the active policy) to a user storage pool, as well as file data that is assigned directly to this storage pool. There is only one default user storage pool; however, you can assign any user storage pool as the default storage pool. See also *user storage pool*

engine
The hardware unit that hosts the software for the metadata server.

event log
A log that maintains a history of event messages issued by all metadata servers in the cluster.

IBM Term: A log that contains information about events for a particular system or group, for a particular metric, or for all the events that are associated with a specific monitor.

failover
The automatic recovery of resources in the event of a network outage, or failure of the hardware or software.

file-placement rule
A rule that controls in what pool SAN File System places files in the global namespace. See also *rule* and *global namespace*.

fileset A hierarchical grouping of files managed as a unit for balancing workload across a cluster.

FlashCopy image
A space-efficient image of the contents of part of the SAN File System at a particular moment.

Forming
A status condition where the cluster has a master and is in the process of forming. This state is always the initial one whenever a cluster is newly formed.

Fully quiescent
A status condition that cuts off all client communication with the cluster.

global namespace
A single file system that provides complete, shared access to both Windows and UNIX clients in the same environment.

ID See *identifier*.

identifier (ID)
A sequence of bits or characters that identifies a user, program, device, or system to another user, program, device, or system.

Initializing
A status condition during which a metadata server or the entire cluster is set up for the first time.

key A property that is used to provide a unique identifier for an instance of a class. Key properties are marked with the Key qualifier. (D)

lease The amount of time that a client can hold a lock.

lock A restriction that allows clients to have exclusive access to files. Types of locks include *data locks*, *session locks*, and *range locks*.

logical unit (LU)
In open systems, a logical disk drive.

logical unit number (LUN)
In the small computer system interface (SCSI) protocol, a unique number used on a SCSI bus to enable it to differentiate between up to sixteen separate devices per SCSI ID address, each of which is a logical unit.

LU See *logical unit*.

LUN See *logical unit number*.

managed object format (MOF)
A compiled language for defining classes and instances. A MOF compiler offers a textual means of adding data to the CIM Object Manager repository. MOF eliminates the need to write code, thus providing a simple and fast technique for modifying the CIM Object Manager repository. (D)

master metadata server
In SAN File System, the metadata server in a cluster that is responsible for physical-space allocation.

metadata
Data that describes the characteristics of stored data; descriptive data.

metadata server
In SAN File System, a server that offloads the metadata processing from the data-storage environment to improve SAN performance. An instance of the SAN File System runs on each engine, and together the metadata servers form a cluster. See also *cluster*.

method
A way to implement a function on a class.

MOF See *managed object format*.

Not running
A status condition where one or more servers in the cluster are not added and therefore the cluster cannot perform any functions.

OBDC
See *one-button data collection*.

object name
An object that consists of a CIM namespace path and a model path. The namespace path provides access to the CIM implementation managed by the CIM agent, and the model path provides navigation within the implementation.

Offline
A status condition during which clients are not being serviced and the cluster is responding only to administrative requests.

one-button data collection
A utility that gathers data for diagnosing errors or failures associated with metadata servers and clients. It is intended primarily for first-failure data-capture capabilities useful for investigating problems upon their initial occurrence, without requiring problem recreation or subsequent tracing.

Online
A status condition that indicates the normal operational state for the cluster.

Partly Quiesced
A state in which the cluster or server is in a "quiet" client communications mode to allow other operations to occur.

Partly quiescent
A status condition that allows existing metadata activity and client communication to continue on the cluster, but prohibits new communication.

policy A list of file-placement rules that define characteristics and placement of files. Several policies can be defined within the configuration, but only one policy is active at one time. See also *file-placement rule* and *service-class rule*.

pool See *storage pool*.

property
An attribute that is used to characterize instances of a class.

qualifier
A value that provides additional information about a class, association, indication, method, method parameter, instance, property, or reference.

quota A limit on the amount of disk space a user can use.

root squashing
The process of a root or Administrator user logging into a client that is not a privileged client.

rule The lines within a policy that specify which actions will occur when certain conditions are met. Conditions include attributes about an object (file name, type or extension, dates, owner, and groups) and the fileset name associated with the object.

SAN File System console
A Web user interface used to monitor and control the SAN File System remotely by using any standard Web browser.

schema
A group of object classes defined for and applicable to a single namespace. Within the CIM agent, the supported schemas are loaded through the managed object format (MOF) compiler.

security log
A log that maintains a history of administrative server login activity.

service location protocol
A directory service that the CIM client application calls to locate the CIMOM.

Shutdown
A status condition that describes when the cluster is shut down as intended.

SLP See *service location protocol*.

Starting
A status condition when a metadata server is starting as designed but is not ready to accept connections from clients.

storage pool
A named set of storage volumes that is the destination for storing client data.

symbolic link
A type of file that contains the path name of and acts as a pointer to another file or directory.

system storage pool
A storage pool that contains the system metadata (system and file attributes, configuration information, and metadata server state) that is accessible to all metadata servers in the cluster. There is only one system storage pool. See also *metadata server*.

user storage pool
An optional storage pool that contains blocks of data that compose the files that are created by SAN File System clients. See also *storage pool* and *default user storage pool*.

volume
A labeled logical unit, which can be a physical device or a logical device. For SAN File System, there is a one to one relationship between volumes and LUNs. See also *logical unit number*.

UTC See *coordinated universal time*

Index

G

I

L

M

N

O

P

Q

R

S

T

U

V

W

Readers' Comments — We'd Like to Hear from You

IBM TotalStorage SAN File System
(based on IBM Storage Tank™ technology)
System Management API Guide and Reference
Version 2 Release 1

Publication No. GA27-4315-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
GA27-4315-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. CGFA
PO Box 12195
Research Triangle Park, NC 27709-9990

Fold and Tape | **Please do not staple** | Fold and Tape

GA27-4315-01

Cut or Fold
Along Line

Printed in USA

GA27-4315-01