ObjectStore®

Managing ObjectStore

Release 6.3

PROGRESS SOFTWARE | *Real Time Division*

Managing ObjectStore

ObjectStore Release 6.3 for all platforms, October 2005

September 2005

Contents

Preface

Purpose

Managing ObjectStore provides information needed to perform management tasks on ObjectStore servers and clients. This book supports ObjectStore Release 6.3.

Audience

There are two audiences for this book:

- Administrators responsible for keeping ObjectStore running and for doing tasks such as backing up and restoring data
- Experienced ObjectStore programmers who need to manipulate the databases they are working with

It is assumed that both audiences are familiar with the ObjectStore host platform and experienced using the operating system.

Scope

Information in this book assumes that ObjectStore is installed and configured.

How This Book Is Organized

The first part of this book provides information that applies to all ObjectStore platforms. Chapters 7 and 8 contain platform-specific information. For complete coverage, you should read the general chapters along with the chapter for your platform.

Notation Conventions

This document uses the following conventions:

Convention	*Meaning*
`Courier`	`Courier font` indicates code, syntax, file names, API names, system output, and the like.
`Bold Courier`	**`Bold Courier font`** is used to emphasize particular code, such as user input.
`Italic Courier`	*`Italic Courier font`* indicates the name of an argument or variable for which you must supply a value.
Sans serif	Sans serif typeface indicates the names of user interface elements such as dialog boxes, buttons, and fields.
Italic serif	In text, *italic serif typeface* indicates the first use of an important term.
`[ ]`	Brackets enclose optional arguments.

Convention	*Meaning*
`{ }*` or `{ }+`	When braces are followed by an asterisk (`*`), the items enclosed by the braces can be repeated `0` or more times; if followed by a plus sign (`+`), one or more times.
`{ a | b | c }`	Braces enclose two or more items. You can specify only one of the enclosed items. Vertical bars represent OR separators. For example, you can specify *a* or *b* or *c*.
`...`	Three consecutive periods can indicate either that material not relevant to the example has been omitted or that the previous item can be repeated.

Progress Software Real Time Division on the World Wide Web

The Progress Software Real Time Division Web site (`www.progress.com/realtime`) provides a variety of useful information about products, news and events, special programs, support, and training opportunities.

Technical Support

To obtain information about purchasing technical support, contact your local sales office listed at `www.progress.com/realtime/techsupport/contact`, or in North America call 1-781-280-4833. When you purchase technical support, the following services are available to you:

- You can send questions to `realtime-support@progress.com`. Remember to include your serial number in the subject of the electronic mail message.
- You can call the Technical Support organization to get help resolving problems. If you are in North America, call 1-781-280-4005. If you are outside North America, refer to the Technical Support Web site at `www.progress.com/realtime/techsupport/contact`.
- You can file a report or question with Technical Support by going to `www.progress.com/realtime/techsupport/techsupport_direct`.
- You can access the Technical Support Web site, which includes
  - A template for submitting a support request. This helps you provide the necessary details, which speeds response time.
  - Solution Knowledge Base that you can browse and query.
  - Online documentation for all products.
  - White papers and short articles about using Real Time Division products.
  - Sample code and examples.
  - The latest versions of products, service packs, and publicly available patches that you can download.
  - Access to a support matrix that lists platform configurations supported by this release.
  - Support policies.
  - Local phone numbers and hours when support personnel can be reached.

Education Services

To learn about standard course offerings and custom workshops, use the Real Time Division education services site (www.progress.com/realtime/services).

If you are in North America, you can call 1-800-477-6473 x4452 to register for classes. If you are outside North America, refer to the Technical Support Web site. For information on current course offerings or pricing, send e-mail to classes@progress.com.

Searchable Documents

In addition to the online documentation that is included with your software distribution, the full set of product documentation is available on the Technical Support Web site at www.progress.com/realtime/techsupport/documentation. The site provides documentation for the most recent release and the previous supported release. Service Pack README files are also included to provide historical context for specific issues. Be sure to check this site for new information or documentation clarifications posted between releases.

Your Comments

Real Time Division product development welcomes your comments about its documentation. Send any product feedback to realtime-support@progress.com. To expedite your documentation feedback, begin the subject with `Doc:`. For example:

```
Subject: Doc: Incorrect message on page 76 of reference manual
```

Chapter 1
Overview of Managing ObjectStore

This chapter briefly describes the architecture of ObjectStore and provides an overview of object-oriented database management tasks. For an introduction to object-oriented database management, including concepts such as persistence, see the first several chapters of the *C++ API User Guide*.

The following topics are discussed in this chapter:

What Is ObjectStore?

ObjectStore is an object-oriented database management system. It allows you to

- Describe, store, and query complex data used in sophisticated computer applications, as well as data traditionally managed by relational database applications, such as MIS programs
- Persistently store data independently of the data type
- Access data in the same format in which it exists in the application
- Create and modify objects instead of tables, columns, rows, and tuples by using the ObjectStore C++, Java, or ActiveX interfaces

What Is an ObjectStore Database?

An ObjectStore database is a storage location for persistent objects. It is organized into clusters and segments.

Clusters

The basic unit of storage allocation in an ObjectStore database is the *cluster*. When you create a persistent object, the storage is allocated from a cluster.

Clusters can be *normal* or *huge*. If an allocation is less than or equal to 64 KB, it is stored in a normal cluster, which can contain any number of normal allocations, up to a maximum size of 2 GB. If an allocation is greater than 64 KB, ObjectStore creates a huge cluster containing just that one allocation.

Segments

Clusters are organized into *segments*. When you create a database, ObjectStore normally creates two segments:

- The *schema segment*, which holds the database roots and schema information about the objects stored in the database. The schema segment cannot be accessed directly by the user application.
- The *default segment*, which stores entities you create with persistent `new`.

You can create additional segments by using the `os_database::create_segment()` function, and you can specify any of these for storage allocation instead of the default segment.

The default segment and any additional segments always have a *default cluster*. ObjectStore normally allocates storage from this cluster. You can create additional clusters by using the `os_segment::create_cluster()` function.

How Do Objects Get into a Database?

When an application allocates an object in persistent storage, it specifies the database to contain that storage. The object is created in the default cluster of the default segment of the specified database. Alternatively, you can specify a segment, and the object is created in the default cluster of the specified segment. Or you can specify a cluster, and the object is created in the specified cluster.

You use the member functions `create()`, `lookup()`, and `open()` of the `os_database` class to create databases. You name the database in a path name argument to the function that creates the database.

See the *C++ API Reference* for information about these functions.

What Kinds of Databases Are There?

You can use ObjectStore to store objects in two kinds of databases: *file databases* and *rawfs databases*. See Managing the Rawfs on page 35 for a discussion of the relative merits of file and rawfs databases.

File Databases

A file database is a native operating system file that contains an ObjectStore database. You can, with some restrictions, manipulate file databases with standard operating system commands as well as with ObjectStore utilities described in this book. A file database has a standard operating system path name.

Rawfs Databases

A rawfs database is a database that you store in an ObjectStore *rawfs*. A rawfs (raw file system) is a private file system managed by the ObjectStore server. It is independent of the file system managed by the operating system.

A rawfs can contain directories, subdirectories, and databases, just as the native file system can. It can include links, but each link must be to another ObjectStore rawfs database or directory either on the same server or on another server. The ObjectStore server manages everything in the rawfs. Everything in the rawfs is invisible to the operating system. ObjectStore provides utilities and `os_dbutil` class functions for operating on databases and directories in a rawfs. An ObjectStore server can manage one rawfs, which consists of one or more rawfs partitions.

Rawfs partitions

On UNIX platforms you specify each partition in the rawfs with a `Partition`*n* statement in the server parameter file. On Windows platforms you specify partitions by using the ObjectStore `setup.exe` utility. Each partition can be either of the following:

- Raw disk space set aside by the operating system, if the operating system supports this. This is referred to as a *raw partition*. Raw partitions have a fixed size.
- A file allocated in the operating system's file system. This is referred to as a *file partition*. File partitions can be expandable or of fixed size.

A rawfs database can span partitions. This allows you to create a database that is larger than any single disk.

When you create a rawfs database, you specify a path name. The server determines where in the rawfs to store your database. Thus, the logical directory structure might

not map directly to the physical placement of the databases in the partitions. For example, two rawfs databases in different ObjectStore directories might be stored in the same partition.

You can create a rawfs during ObjectStore installation or at any time after installation. See Chapter 7, Managing ObjectStore on UNIX, on page 299, or Chapter 8, Managing ObjectStore on Windows, on page 317, for more information about how to create a rawfs.

Rawfs database name format

Use the following name format to specify an ObjectStore rawfs database:

hostname`::`*/database_pathname*

A double colon separates the host name from the database path name. The name of a rawfs database always starts at root; it is never relative to a working directory. Slashes always separate the levels of a rawfs database name, regardless of the platform. Case is significant. For example, the following two path names identify different databases:

```
hostess::/accounts/payable/JUNE
hostess::/accounts/payable/june
```

How ObjectStore Controls Storage

The server is the ObjectStore process that primarily controls object storage. With help from the client process and the cache manager process, the server can manage databases for multiple client applications. These applications can be on one or multiple hosts.

What Does the Server Do?

The ObjectStore server is a process that controls access to ObjectStore databases on a host. This includes

- Storage and retrieval of persistent data
- Arbitration of concurrent access by multiple client applications
- Recovery of databases to a transaction-consistent state if any process aborts or any host crashes, or in the event of network failure

The server also manages pages of data on behalf of clients running applications.

For the rawfs, if there is one on the host, the server manages the hierarchy of directories and maintains permission modes, creation dates, owners, and groups for each entry.

Typically, a server must be running before any ObjectStore application can access databases on the host. (A locator file allows access to databases residing on a host that is not running a server. See Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 265, for further information.)

An application can use databases that are stored on different hosts and managed by different servers. A server can serve clients on any number of hosts. A network can have a number of servers.

Multiple servers on a host

More than one ObjectStore server can run on the same host. If you want to run two servers of different releases on the same machine (for example, Release 6 and Release 5), you must start them on different ports and use the ports file to let clients know the one to contact. See Modifying Network Port Settings on page 58 for details. Note that a single client cannot address multiple servers running different releases of ObjectStore.

Under ObjectStore Release 6.1 and later, a client can address two or more servers of the same release, running on the same machine. This feature is chiefly useful when enabling failover (see Failover on page 285). It allows you to configure ObjectStore so that, in a failover situation, the services of the failed server can be restored on a server that is running on a second machine, even if another server was previously running on that machine. In this situation, a client can address both the previously running server and the failover server. For more information, see "Configuring more than One Logical Server" on page 294. Note that this feature is only available on ObjectStore Release 6.1 or later.

What Does the Client Application Do?

ObjectStore links the client library into each ObjectStore application. In this way, each ObjectStore application is an ObjectStore client that

- Maps persistent database objects to virtual addresses
- Allocates and deallocates storage for persistent objects
- Maintains the cache of recently used pages and the lock status of those pages
- Handles page faults on addresses that refer to persistent objects

Any number of clients can run on a particular host. These clients can contact

- The server on that host
- Any other server on any other host in the network

Client cache

Each client has its own storage area, called the *client cache* or simply the *cache*. The cache is a local holding area for data mapped or waiting to be mapped into virtual memory. Caches are not shared by clients even when they are running on the same host.

To change the default size of the client cache, use the `OS_CACHE_SIZE` environment variable. See `OS_CACHE_SIZE` on page 106.

When a client application needs an object stored in a database, it dereferences a pointer (or object reference in Java). If the page that holds the object is already in physical memory or if the page is in the cache with the appropriate permission, access is immediate.

If the page that holds the object is not in the cache, or is in the cache but with inappropriate permission, the application takes a page fault. This causes the client to

request the page from the server, put it in the cache, and resume executing the program.

UNIX

On UNIX platforms, by default, ObjectStore places the cache file in the `/tmp/ostore` directory. To change the default, specify an alternate directory for the `OS_CACHE_DIR` environment variable, or set the `Cache Directory` parameter in `$OS_ROOTDIR/etc/`*`host`*`_cache_manager_parameters`.

Windows

On Windows platforms, the operating system determines the location of the cache in virtual memory. You cannot change the location. The cache is not a file; it is a region of virtual memory. The necessary storage is obtained from system virtual memory, which consists of physical memory plus swap file space.

What Does the Cache Manager Do?

The primary function of the cache manager is to facilitate concurrent access to data by handling callback messages from the server to client applications. Note that the cache manager never reads the cache itself. The cache manager coordinates access by clients to cached data.

A cache manager starts automatically when an ObjectStore application starts, if a cache manager is not already running on the host. Each host that runs an ObjectStore application must have one cache manager. A single cache manager can handle callback messages for any number of client applications running on that host.

If you are running clients from two different major releases of ObjectStore, there are two cache managers — one for each release. Unlike running different servers on one host, you need not configure ports.

The name of the cache manager executable is `oscmgr6`.

On UNIX, the `oscminit6` executable starts `oscmgr6`. Typically, you never need to invoke `oscminit6`.

Callbacks

When a client requests permission to read a page and no other client has permission to modify that page, the server grants read permission (read *ownership*). The cache manager is not involved.

The cache manager is involved in the following situations:

- When a client requests permission to read or modify a page and another client has permission to modify that page
- When a client requests permission to modify a page and other clients have permission to read that page

In these situations, the clients with permission are blocking the requesting client from obtaining permission, so the server sends a callback message to the cache manager on the host of the client that has the permission.

The server cannot send callback messages directly to the client because the client might not be listening; the client might be busy running the application. The cache manager determines whether the read or write permission can be released or whether the client requesting permission must wait.

If you are running an ObjectStore application that uses a database that no other ObjectStore application is accessing, there are no callback messages for that database.

For information about ownership and locks, see Ownership and Locks on page 64.

Commseg

The *commseg* (communications segment) is a shared-memory object where the client and the cache manager maintain information about permissions on pages and about whether the client is actually using the page. Each client has its own commseg. For every page in the cache, there is a corresponding item in the commseg.

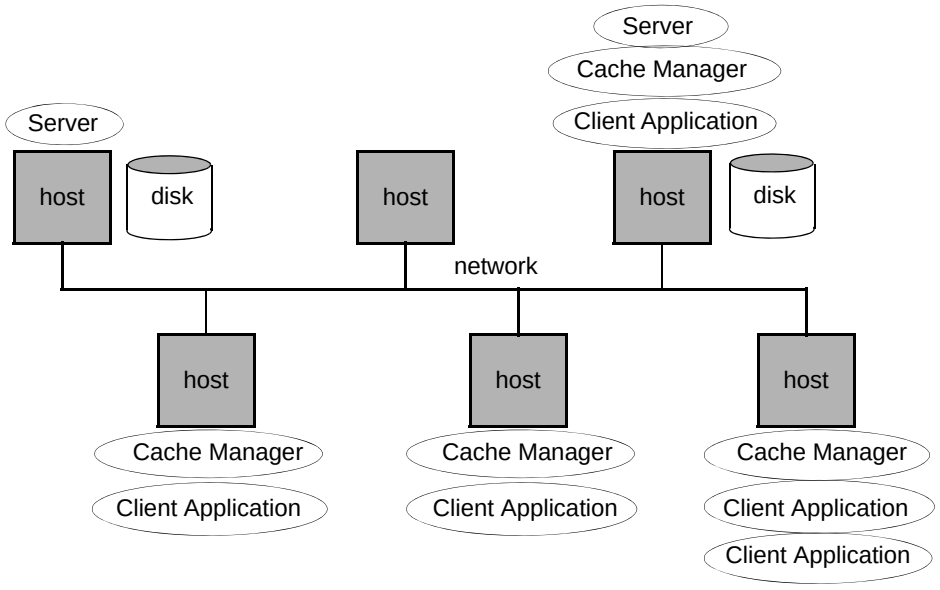
UNIX

On UNIX platforms, by default, ObjectStore places the commseg file in the `/tmp/ostore` directory. You can specify an alternative directory by setting the `OS_COMMSEG_DIR` environment variable. Another way to change the default is to set the `Commseg Directory` parameter in the `$OS_ROOTDIR/etc/`*`host`*`_cache_manager_parameters` file.

Windows

On Windows platforms, the operating system determines the location of the commseg in shared memory. The commseg is not a file; it is a region of virtual memory.

The following illustration shows one possible configuration of ObjectStore processes.



Managing Processes

ObjectStore includes three main processes that communicate with each other to manage your data:

- Server (`osserver`)
- Client (application)

- Cache manager (`oscmgr6`)

In general, the actions you perform on ObjectStore processes are

- Starting up
- Shutting down
- Obtaining process-status information

Communication Among ObjectStore Processes

The following table summarizes the communication among the server, client, and cache manager processes.

Process	*Description*
Server	Responds to client requests for pages
	Sends callback messages to cache manager to request release of ownership held by a client
Client	Requests the server to fetch data from a database
	Requests the server to store data in a database
	Receives locks and pages from the server
Cache manager	Receives callback requests from the server
	Creates client cache and commseg files

ObjectStore uses network connections to communicate among server, client, and cache manager processes. The kind of network connection used depends on your platform. Typically, you need not modify network connections. However, if you do need to make changes, see Modifying Network Port Settings on page 58.

Starting ObjectStore Processes

Server

Chapter 7, Managing ObjectStore on UNIX, on page 299, or Chapter 8, Managing ObjectStore on Windows, on page 317, provides instructions for starting the server on your platform. Typically, the installation procedure arranges for the server to be started automatically when the system is booted.

When you start a server, the server checks for values of server parameters you might have changed from the default and uses the modified values. If you have not modified any server parameters, ObjectStore uses the default parameters.

The server then makes its service available. A client can use the network connection available on its platform to connect to the server. ObjectStore uses default network connections. To modify these connections, see Modifying Network Port Settings on page 58.

There are many server parameters you can set to determine the behavior of the server. Chapter 2, Server Parameters, on page 77, describes each parameter. When you modify a parameter, you must shut down and restart the server for the parameter to take effect.

Client — When a client application starts, it tries to connect to the cache manager on that machine. If a cache manager is not running, ObjectStore starts one.

Cache manager — The cache manager starts automatically if one is not already running when a client application starts.

Windows — On Windows platforms, the server and cache manager typically run as Services. You can use the Services applet in the Control Panel to start and stop the server and cache manager and to determine whether they start automatically when you boot the system.

Stopping ObjectStore Processes

Server — You need to shut down the server in the following situations:

- Before you reboot or halt the host
- After you modify server parameters
- To resize the transaction log (see Log File Size on page 28)
- To add a partition to the rawfs

Follow these steps to shut down the server:

1. Use the `ossvrstat` utility to determine whether clients are using the server.

   Along with other information, this utility displays client names if they have been set. To identify clients easily, encourage developers to use the `objectstore` class, `set_client_name()` function. See ossvrstat on page 248.

2. Notify clients to end their connections with the server.
3. Use the `ossvrclntkill` utility to end the server's connection with any *dangling clients*.

   Dangling clients are clients still attached to the server even though they no longer exist. This can happen when a client is halted abnormally rather than being stopped in the usual manner. See ossvrclntkill on page 242.

4. Use the `ossvrshtd` utility to shut down the server. See ossvrshtd on page 246.

For information about shutting down a server running on a node of the Sun Clusters operating system, see Cluster operating system on page 246.

To restart the server, see the instructions in the chapter supporting your platform.

Client — Shutdown of a client is the responsibility of the application.

If necessary, you can use the `ossvrclntkill` utility to sever the connection between a client and the server. This disconnects the client. See ossvrclntkill on page 242 for details.

Cache manager — Before you shut down the cache manager, notify clients that you are shutting it down and then use the `oscmstat` utility to confirm that there are no active clients. See oscmstat on page 157. Use the `oscmshtd` utility to shut down the cache manager. See oscmshtd on page 156 for details. The next client process that starts automatically starts the cache manager.

Obtaining Process-Status Information

ObjectStore provides a variety of functions to obtain information about the state of the server and clients. Some types of information are generated automatically; for others you run an ObjectStore utility to gather the information.

Process Messages

When an ObjectStore daemon process sends output to `stdout` or `stderr`, ObjectStore routes the output to a corresponding file. These files have different names on different platforms. For UNIX, see Finding Files Containing ObjectStore Messages on page 313 in Chapter 7; for Windows, see Finding Files Containing ObjectStore Messages on page 322 in Chapter 8.

ObjectStore daemons seldom send messages to these files except in debug mode or under certain unusual error conditions. In these cases, this information can be helpful in understanding and resolving an error. In debug mode, the ObjectStore server and cache manager send a lot of output to these files because all of the debugging output goes there.

When you report a problem involving one of the ObjectStore daemons to Technical Support, find such a file if it exists and provide the contents.

Displaying Status Information

The `ossvrstat` utility displays information from various server meters. Examples of this information are

- The amount of data the server sent to clients
- The amount of modified data clients sent to the server
- The number of committed transactions the server knows about
- The number of times the server chose a deadlock victim
- The number of times a message from a client to the server had to wait to use a message buffer
- The amount of data in the log

This utility provides these meters and many more for several periods of time, such as the last hour and the last minute. See ossvrstat on page 248.

The `oscmstat` utility displays information about the cache manager, client cache files, and commseg files. See oscmstat on page 157.

Debugging the Cache Manager

Use the following command line to start the cache manager in debug mode:

```
oscmgr6 -d debug_level
```

For *`debug_level`*, specify an integer from 0 through 50. The higher the number, the more information ObjectStore displays about cache manager activity.

Pinging

If you are having problems with a server, the first thing to do is run `ossvrping` to see if the server is running. See ossvrping on page 245.

Monitoring the Server

You can run the server in debug mode to obtain information about exactly what is happening. ObjectStore displays messages about process communication that show where the problems are, if there are any.

There are two ways to enable debug mode:

- Shut down and restart the server with the debug option. See Chapter 7, Managing ObjectStore on UNIX, on page 299, or Chapter 8, Managing ObjectStore on Windows, on page 317, for specific details for your platform. When you no longer need the debug output, shut down the server and restart it without the debug option. Running the server in debug mode slows down the server but does not affect clients. Use debug mode only when you are experiencing a problem.
- If you want to turn debug mode on without shutting down the server, you can use the `ossvrdebug` utility to set the debug trace level for the server. To enable server debugging, type `ossvrdebug -d` *`debug-level`* where *`debug-level`* is an integer from 0 to 9. To disable server debugging, type `ossvrdebug -d 0`. See ossvrdebug on page 244 for more information.

The Server Transaction Log

Each server keeps a *transaction log*, also called a *log file*. The most important function of the transaction log is to prevent database corruption in case of failure. The log contains modified pages of data and the records about modified pages of data.

The log file stores database modifications until they are propagated to the database. The log does not have a fixed size. It can grow to contain as much data as is written to it. You can use the `ossvrstat` utility to obtain the size of the log. This is described more fully in ossvrstat on page 248.

By default, the server places its log file in the rawfs. If there is no rawfs, you must use the `Log File` server parameter to specify a path name for the transaction log. When you use the `Log File` server parameter, you cannot place the log file in a raw partition. See `Log File` on page 92.

When the log is in the rawfs, it is not visible to you. Its size is limited by the size of the rawfs. The log can span partitions.

When the log is in the native file system, its size is limited by the file partition size. You should place the log file where it can never be deleted by users accidentally. Deleting the log file can cause database corruption.

For best performance, the log file should be on a disk that is not used for anything else.

When a transaction modifies databases, either all or none of the modifications are made, depending on whether ObjectStore commits or aborts the transaction. This is true even if the server machine crashes during the transaction.

If your transaction aborts, the log entries are discarded. When your transaction commits, your program waits until the log information is safely on disk.

The server moves committed data from the log to databases through *propagation*. The information accumulated in the log is propagated from the log to the material (that is, real) database transparently in a way that does not interfere with client performance. After propagation, the space in the log that was occupied by propagated data becomes available for new entries in the log.

Log File Size

When the server recognizes that it needs a bigger transaction log, it increases the size of the log segments according to the `Log Data Segment Growth Increment` and `Log Record Segment Growth Increment` server parameters. See `Log Data Segment Growth Increment` on page 91 for details.

When you run the `ossvrstat` utility, `Current log size`, in the list of server parameters, displays the current size of the log in terms of the number of *sectors* involved. A *sector* is a 512-byte disk block (two sectors equal one kilobyte). Usually, the log grows to a size that accommodates your application and then stops growing. There usually is no reason to monitor log size or to worry about its allocation. However, if an unusual event causes the log to grow too large, there are ways to make it smaller.

If there is nothing in the log when you start the server, the server resets the size of the log data segment and log record segments to the sizes set by the `Log Data Segment Initial Size` and `Log Record Segment Initial Size` server parameters. For log files in the rawfs, this means the space is free for other databases. For log files in the native file system, this reallocates internal log file space; the log file size does not change.

Shrinking the log file

To move or shrink the log, follow these steps:

1. Set the `Log Data Segment Initial Size` and `Log Record Segment Initial Size` server parameters to the values you want. If necessary, ensure that the correct value is specified for the `Log File` server parameter.
2. Ensure that no clients are using the server.
3. Run `ossvrshtd` to shut down the server.

   The server propagates the log before it actually shuts down. For information about shutting down a server running on a node of the Sun Clusters operating system, see Cluster operating system on page 246.
4. If you are reallocating a log in the rawfs, skip to step 5.

   If you are reallocating a log in the native file system, run the server executable with the `-ReallocateLog` option.

   This does not actually start the server; it only reallocates the log. Enter

```
osserver -ReallocateLog
```

5 Restart the server. For more information, see Starting the Server on page 302 in Chapter 7 for UNIX platforms or Starting the Server on page 319 in Chapter 8 for Windows platforms.

Warning

Never use `-ReallocateLog` if the server is in an inconsistent state. Doing so can cause database corruption. Call Technical Support for assistance if necessary.

Managing Computer Resources

The computer resources you must manage are your CPUs, memory, disk space, and network.

CPUs

Consider the speed of your CPUs. You can run computation-intensive applications on your fastest machines, or you can distribute intensive applications evenly among network hosts. You can also combine these approaches.

Memory

ObjectStore uses several kinds of memory. Managing Memory on page 30 describes effective memory management.

Disk space and network

Network communications and disk input and output often are the most constraining elements of your installation. Configure your system to minimize network and disk activities. Ensure that applications are designed to minimize this activity. Also, you probably want most of your disks, especially your fastest disks, on the server.

Predicting the amount of disk space you need for a client application requires a good understanding of the particular application. In general, however, an ObjectStore application uses disk space as follows:

- On UNIX systems, the default location for the cache and commseg files is `/tmp/ostore`. On Windows, the cache is in virtual memory and the commseg is in shared memory. The operating system determines both locations. If necessary, you can increase the amount of virtual address space by configuring a larger swap file.
- ObjectStore uses swap space for transient data when it cannot be mapped into physical memory during a transaction. See Managing Memory on page 30.

The following table shows how heavily ObjectStore uses resources:

Process	*CPU Use*	*Disk Use*	*Network Use*	*Memory Use*
Server	Not intensive.	Intensive.	Intensive.	Not intensive. Uses a little to communicate with client.

Process	*CPU Use*	*Disk Use*	*Network Use*	*Memory Use*
Client	Can be intensive; depends on application.	ObjectStore does not use local disk. Application might.	Can be intensive.	Can be highly intensive. Lots of mapping of data in virtual memory.
Cache manager	Not intensive.	Not intensive.	Not intensive. Sends and receives short messages.	Not intensive.

Managing Memory

An ObjectStore application uses three kinds of memory:

- Physical memory is the real memory (RAM) available on the machine. All applications running on the machine share the real memory
- Virtual memory contains the application's data. Each application has its own virtual memory.
- Persistent memory is that in which you can store an application's persistent data. Each application has its own persistent memory.

  Persistent memory is not a limitation on the amount of persistent data an application can have. It is just the place in which the ObjectStore client manipulates persistent data for a transaction. Persistent memory is a subset of virtual memory.

To manage the memory that an ObjectStore application uses, you need to understand the concept of *address space*.

What Is Address Space?

ObjectStore transfers an application's data to virtual memory. Data stored in virtual memory can, at any particular time, reside in

- Physical memory
- The client cache, which is the backing store for persistent data
- Swap space, which the backing store for transient data

An application's virtual memory contains both

- Persistent data that the application accesses in an ObjectStore transaction
- Transient data that is static or allocated in the stack or heap

Address space is a range of virtual memory addresses. For most 32-bit computers, the address space is slightly less than 2^{32} bytes. (The platform's virtual memory system might reserve or might not implement a portion of the 2^{32} range.) For 64-bit computers, the address space is substantially larger.

Each client process has its own address space. Address space is distinct from virtual memory in that

- Address space includes all addresses that could be assigned. It does not matter whether the address is in use. It typically is larger than virtual memory.
- Virtual memory includes only addresses that currently contain application data, either in physical memory or in backing store on disk.

How Are the Pieces of Address Space Shared?

A client's address space includes two kinds of addresses: persistent and transient. The data itself can be in one of three locations in physical memory or backing store. These are described in detail in this section.

Kinds of addresses

Address space includes

- Persistent address space — This is a range of addresses that ObjectStore uses to store only persistent data.
- Transient address space — This includes all addresses in the address space except those designated for persistent data.

Just as each client has its own address space, each client has its own persistent address space, called the *persistent storage region* or *PSR*. Two environment variables control the size and address range of the PSR. `OS_AS_SIZE` and `OS_AS_START` have different default values on different platforms. For example, the PSR on a SPARCstation 10 running Solaris 2 typically is 400 MB. See `OS_AS_SIZE` on page 103 and `OS_AS_START` on page 104 for specific details.

Each client also has its own transient address space. This space includes all addresses not part of the PSR.

Locations for data

An application's data is stored in virtual memory. This means that data is in one of the following:

- Physical memory
- Backing store
  - Client cache for persistent data
  - Swap space for transient data

Physical memory is shared by all applications running on that machine. A typical amount of physical memory for a machine is 32 to 256 MB.

When a page of data needs to be brought into physical memory, the operating system often needs to make room by removing another page. When the operating system removes a page from physical memory, it places the page on disk in the appropriate backing store.

If the page contains persistent data, the operating system pages it to the client cache. Each client has its own cache.

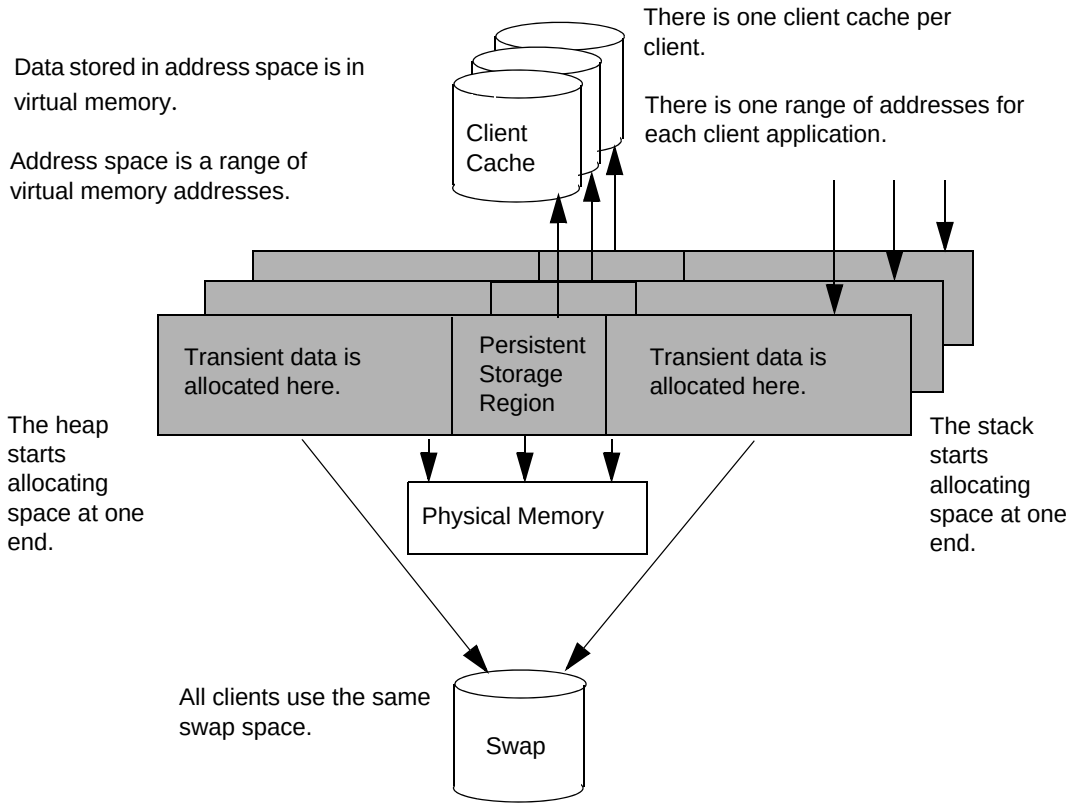
If the page contains transient data, the operating system pages it to *swap space*. Swap space is a disk file or disk partition that is shared by all applications running on the host.

Virtual memory

Remember that data in an application's address space is always in virtual memory. It does not matter whether the data has a persistent address or a transient address. It also does not matter whether the data is in physical memory or backing store.

Illustration of Address Space

The following figure shows address space. The persistent storage region is near the middle of the address space. The stack allocates transient data starting at one end of the range of addresses. The heap allocates transient data starting at the other end of the range of addresses. The stack and the heap can allocate space until they reach the persistent storage region. However, on some platforms the stack and heap can grow large enough so that they infringe on the PSR. This situation may cause your application to suddenly contain invalid pointers. It can also cause ObjectStore to fail to initialize correctly or to send `mmap failed` errors, indicating it could not allocate memory. To avoid or troubleshoot this situation, conatct ObjectStore Technical Support for information about tools you can run to monitor address space use.



How Do Transactions Use Address Space?

The PSR limits the amount of data you can touch within a single top-level transaction. Address space in the PSR must be reserved to correctly relocate data into

virtual memory. Space is reserved as each page is used in the transaction, and the amount of space is the minimum required to relocate the page correctly into virtual memory.

Normally, after a top-level transaction commits or aborts, the assignments of persistent space are made available for reuse.

However, if the application is using `objectstore::retain_persistent_addresses()`, ObjectStore does not release persistent address space until the program calls `objectstore::release_persistent_addresses()`.

If a transaction commits, the client

- Sends a copy of each modified page back to the server
- Keeps a copy of each modified page in the cache in case the page is needed in the next transaction
- Keeps in the cache those pages already in the cache but not modified

If a transaction aborts, the client

- Throws away modified pages because they are no longer valid
- Keeps in the cache those pages already in the cache but not modified, in case they are needed in the next transaction

An application can limit the amount of address space it uses by

- Limiting the amount of persistent dara that is accessed by a transaction. An effective way to do this is to cluster the data in a transaction's working set so that it uses the minimum number of pages.
- Using ObjectStore soft pointers rather than hard pointers. ObjectStore reserves address space for objects that are pointed to, even if the transaction does not touch these objects. But it does not reserve address space for objects referenced by soft pointers until they are dereferenced. See Using ObjectStore Soft Pointers in Chapter 1 of the *Advanced C++ API User Guide* for more information.

What Happens When Resources Are Exhausted?

Address resources are large, and it is unusual to exhaust them. However, lack of a particular resource might affect an application.

Persistent address space

If persistent address space is needed but is not available, ObjectStore raises the `err_address_space_full` exception. This can happen when a transaction tries to assign to the persistent storage region more data than the PSR can accommodate.

Physical memory

When physical memory is needed but is not available, the operating system swaps pages to backing store. The application continues to run; but if a great deal of swapping (thrashing) occurs, program execution is slower.

Client cache

When space in the client cache is needed but is not available, ObjectStore attempts to migrate modified pages from the client cache back to the server that supplied the data. This can also degrade performance.

Swap space

If swap space is needed but is not available, the operating system does one or both of the following:

- Prohibits other processes from starting.
- Aborts the process that required additional swap space.

  Often a process is aborted when it tries to allocate more memory; the allocation function returns `0`. If an ObjectStore allocation fails in this way, `err_insufficient_virtual_memory` is signaled.

How Can You Control These Resources?

Persistent address space

Increasing the size of the persistent storage region makes more address space available for a client's persistent data. This allows a transaction to access more persistent data, which in turn might change a transaction that aborts to a transaction that commits.

The environment variables `OS_AS_START` and `OS_AS_SIZE` control the amount of persistent address space. Each application can use the default values or specify its own settings. See `OS_AS_SIZE` on page 103.

Physical memory

You can add physical memory to the machine to lower the reliance on backing store. This increases the performance of all applications (ObjectStore as well as non-ObjectStore) because swapping pages out of physical memory happens less often. Operating systems include tools that help you determine when an application is paging; for example, `time` and `top` in UNIX or Performance Monitor in Windows.

Client cache

Increasing the size of the cache can improve performance if it decreases the need to send pages containing persistent data back to the server. However, in an application that operates on large datasets, a larger cache may mean more swapping of pages, which will lower performance. You control the size of the client cache with the `OS_CACHE_SIZE` environment variable. See `OS_CACHE_SIZE` on page 106.

Swap space

Adding more swap space allows you to run more or larger applications. You can add swap space by following operating system-specific procedures. Some operating systems require a system reboot to accomplish this, while others allow swap space to be added while the machine is running. Your operating system includes a tool for determining the amount of swap space available, for example, `pstat -s` in UNIX or `the` System icon in the Windows Control Panel. Increasing the size of the client cache or adding swap space provides more potential virtual memory for the client's data.

How Much Memory Is Needed?

The amount of memory your application needs depends entirely on the application. Work with the application developers to understand the application's data structures and data access patterns. Start by using the default settings for the variables that control address space. Use test runs of the application to refine memory allocation.

Reserving Versus Mapping an Address

When considering the way a client uses address space, it is important to understand the difference between *reserving* and *mapping* an address.

When ObjectStore reserves an address for a page, it determines where to put the page if the client needs it. The data on the page is not available to the client. Reserving an address ensures that the address cannot be assigned to another page. ObjectStore assigns address space in units of 64 KB, regardless of the page size on the platform. Reserving an address consumes address space but not physical memory or swap space.

When ObjectStore maps a page to an address, it means that the page is available to the client. The client can now use the data on that page. Mapping occurs one page at a time and consumes virtual memory (physical memory and, perhaps, swap space).

Managing the Rawfs

This section presents general information on managing a rawfs in an ObjectStore environment. See Chapter 7, Managing ObjectStore on UNIX, on page 299, or Chapter 8, Managing ObjectStore on Windows, on page 317, for more platform-specific information on creating a rawfs.

Advantages

The advantages of a rawfs over an environment that uses operating system file databases are that it

- Allows for large databases because databases can span partitions. Information in a single database is not limited to the maximum size of a file, which on some operating systems is limited to 2 or 4 GB. (*Large* usually refers to multigigabyte databases, but it also depends on the amount of disk space available.)
- Allows applications to use ObjectStore in such a way that ObjectStore is invisible to end users.
- Simplifies management because all databases are in a single location.

Security

A rawfs can provide greater security because the

- Transaction log is hidden, so accidental deletion cannot occur.
- Content of the rawfs is accessible only through ObjectStore. The rawfs is not visible to the operating system.
- Database contents are protected at the segment level.

The disadvantages of a rawfs are that it

- Has a fixed size. You must explicitly add a partition to increase the size. While file partitions in a rawfs can be expandable in size, the performance for file partitions is sometimes not as good as it is for raw partitions.
- Cannot be accessed with operating system commands, so you lose the flexibility offered by those commands.
- Might be difficult to back up because of its size.

Utilities for Managing the Rawfs

ObjectStore provides the following utilities for managing the rawfs. You can use many other utilities on the rawfs and on rawfs databases, but these supply functionality that would otherwise be provided by the native operating system.

ObjectStore utility	*Function*
`oschgrp` on page 150	Changes a database group name
`oschhost` on page 151	Changes link hosts
`oschmod` on page 152	Changes database permissions
`oschown` on page 154	Changes database owner
`oscopy` on page 167	Copies a database
`osdf` on page 172	Displays used and available disk space
`osln` on page 190	Creates a link
`osls` on page 194	Lists a directory
`osmv` on page 196	Moves a directory, database, or link
`osrm` on page 213	Removes a database or link
`osrmdir` on page 214	Removes a directory
`ostest` on page 257	Tests a path name for specified conditions

Note that these utilities can also be used on the file system supported by the native operating system. In this case, many of the ObjectStore utilities invoke the corresponding tool provided by the operating system.

ObjectStore utilities that operate on rawfs directories and databases can perform wildcard processing. The wildcards you can use are described below. For example, to list all databases in the `sax` directory that start with `charlie,` you enter the following:

```
osls oscar::/sax/charlie*
```

The following table describes the wildcard characters you can use:

Wildcard	*Description*
`*`	Matches any string, including a null string.
`?`	Matches any single character.
`[...]`	Matches any one of the enclosed characters. A pair of characters separated by a dash (–) matches any character lexically between the pair, inclusive. If the first character following the opening bracket (`[`) is an exclamation point (`!`), any character not enclosed is matched.
`{...}`	Matches any one of the enclosed sequences. For example, `file.{cc,hh}` matches `file.cc` and `file.hh`.

UNIX

On UNIX systems, you must precede the wildcard with a backslash (\) or enclose the path name with the wildcard in quotation marks (" "). This prevents the shell from interpreting the wildcard as a shell wildcard.

Reconfiguring the Rawfs

You need to reconfigure a rawfs if you want to shrink the size of the rawfs to save space or you want to remove a partition. Reconfiguring a rawfs involves wiping out the existing rawfs.

See Chapter 7, Managing ObjectStore on UNIX, on page 299, or Chapter 8, Managing ObjectStore on Windows, on page 317, for specific instructions about creating a rawfs and starting the server on your platform. If you decide to reconfigure the rawfs, use that information with these general instructions.

To reconfigure a rawfs:

1. Confirm that you are on the machine you want to reinitialize and that the `OS_ROOTDIR` environment variable reflects this. You must be careful not to delete another rawfs accidentally.
2. Use the `ossvrstat` utility to confirm that no users are changing any databases in the rawfs. The rawfs databases are backed up for the last time during this procedure.
3. Make sure that no clients attempt to use databases during the rest of this procedure.
4. Use the `ossvrchkpt` utility to move everything in the log to the relevant databases.
5. Use the `osbackup` or `oscopy` utility to back up the databases in your rawfs.
6. Use the `ossvrshtd` utility to shut down the server. For information about shutting down a server running on a node of the Sun Clusters operating system, see Cluster operating system on page 246.
7. On UNIX platforms, modify the `PartitionN` server parameter specifications to reflect your new rawfs. On Windows platforms, use the ObjectStore `setup.exe` utility to modify the configuration of the rawfs.
8. Initialize the server by specifying `osserver -i`. This wipes out the rawfs partitions.
9. Start the server.
10. Use the `osrestore` utility (or `oscopy`, if you used it when saving the database) to restore any necessary databases.

What You Need to Know About the API

Some administrative tasks require familiarity with aspects of the ObjectStore API. The following sections discuss these aspects of the API.

Capacity Planning

Capacity planning requires knowledge of the data structures underlying any ObjectStore classes that the application stores, such as collection, relationship, and reference classes. See the following related ObjectStore documentation.

For Information About	*See*
Soft pointers	Chapter 1, Advanced Persistence, in the *Advanced C++ API User Guide*
Collections	Chapter 1, Introducing Collections, in the *C++ Collections Guide and Reference*
Relationships	Chapter 4, Data Integrity, in the *Advanced C++ API User Guide*

Data Organization

Managing the physical organization of the data (what is stored next to what) requires knowledge of the application's creation and deletion patterns and clustering directives. For information about creation and deletion patterns, talk to the application developers. For information about clustering, see Chapter 3, Programming with Persistent Storage, in the *C++ API User Guide*.

Indexes and Contention

Managing indexes uses the API, and managing contention sometimes involves the API (for example, controlling locking granularity and lock caching). For information about indexes, see Chapter 7, Using Indexes to Optimize Performance, in the *C++ Collections Guide and Reference*. For information about controlling locking granularity and lock caching, see Chapter 2, Advanced Transactions, in the *Advanced C++ API User Guide*.

Managing ObjectStore Programmatically

ObjectStore provides various utilities for performing different routine administrative tasks. For example, the `osverifydb` utility can be used to verify pointers and references in an ObjectStore database.

Although the utilities are intended for use on the command line, their functionality can also be invoked programmatically. The ObjectStore API includes the `os_dbutil` class whose member functions correspond to many of the utilities. For example, pointers and references in a database can also be verified by calling the `os_dbutil::osverifydb()` function within a program. The `os_dbutil` class thus enables you to use its member functions as the basis for writing your own utilities.

The following tables list the different administrative tasks you can perform programmatically by calling member functions of the `os_dbutil` class. Note that before using any of these functions, you must call `os_dbutil::initialize()`. For reference information about `os_dbutil` and its functions, see `os_dbutil` in Chapter 2 of *C++ API Reference*.

The tables are organized according to the following components of ObjectStore that might require management:

- Server
- Client
- Cache manager
- Database
- Schema

Managing Servers

Task	*Function*
Determining sector size	`os_dbutil::get_sector_size()`
Making the server take a checkpoint	`os_dbutil::svr_checkpoint()`
Killing a client thread on a server	`os_dbutil::svr_client_kill()`
Enabling debugging information about the server	`os_dbutil::svr_debug()`
Mapping a server to a host	`os_dbutil::svr_machine()`
Pinging a server to determine if it is running	`os_dbutil::svr_ping()`
Shutting down a server	`os_dbutil::svr_shutdown()`
Getting information about the server	`os_dbutil::svr_stat()`

Managing Clients

Task	*Function*

Closing client connections	`os_dbutil::close_all_connections()`
	`os_dbutil::close_all_server_connections()`
	`os_dbutil::close_server_connection()`

Getting a client name	`os_dbutil::get_client_name()`
Setting a client name	`os_dbutil::set_client_name()`

Managing Cache Managers

Task	*Function*
Removing unused cache and commseg files	`os_dbutil::cmgr_remove_file()`
Shutting down the cache manager	`os_dbutil::cmgr_shutdown()`
Getting information about the cache manager	`os_dbutil::cmgr_stat()`

Managing Databases

Task	*Function*
Copying a database	`os_dbutil::copy_database()`
Getting the host name of a database	`os_dbutil::hostof()`
Moving databases offline or online	`os_dbutil::osdbcontrol()`
Getting the size of a database	`os_dbutil::ossize()`
Verifying pointers and references in a database	`os_dbutil::osverifydb()` `os_dbutil::osverifydb_one_segment()`
Controlling output messages during database replication	`os_dbutil::start_backrest_logging()` `os_dbutil::stop_backrest_logging()`

Managing the Rawfs

Task	*Function*
Changing the primary group of a directory or rawfs database	`os_dbutil::chgrp()`
Changing ownership of databases and directories	`os_dbutil::chown()`
Changing database permissions	`os_dbutil::chmod()`
Getting rawfs disk space information	`os_dbutil::disk_free()`
Expanding a rawfs file name	`os_dbutil::expand_global()`
Testing for a rawfs pathname	`os_dbutil::file_db_pathname()` `os_dbutil::split_pathname()`
Listing the contents of a rawfs directory	`os_dbutil::list_directory()`
Making a rawfs directory	`os_dbutil::mkdir()`
Moving a rawfs directory or database	`os_dbutil::rename()`
Changing the host referenced by a link in the rawfs	`os_dbutil::rehost_link()` `os_dbutil::rehost_all_links()`
Removing a database or rawfs link	`os_dbutil::remove()`
Moving a rawfs directory or database	`os_dbutil::rename()`
Removing a rawfs directory	`os_dbutil::rmdir()`
Getting information about a rawfs pathname	`os_dbutil::stat()`

Managing Schema

Task	*Function*
Comparing schemas	`os_dbutil::compare_schemas()`
Setting the path of the application schema database	`os_dbutil::set_application_schema_path()`

Managing Databases

When managing an ObjectStore database, you are likely to perform the following operations:

Operation	*Utility and Reference*
Copy	`oscopy` on page 167
Move	`osmv` on page 196
Delete	`osrm` on page 213
Backup	`osbackup` on page 143
Restore	`osrestore` on page 208
Dump	`osdump` on page 173
Load	`osload` on page 192
Compact	`oscompact` on page 160
Verify	`osverifydb` on page 258

For information about these and the other utilities for managing ObjectStore, see Chapter 4, Utilities, on page 131.

Working with Databases and Data Sets

Database management is complicated by the fact that a database often points to other databases and is pointed to by other databases. Databases that have such relationships constitute a *data set*. To perform many administrative activities, you must identify the databases that make up the data set. For example, a database called `databaseA` might be part of a data set that includes

- `databasea`
- All databases that point to or reference `databaseA` and their data sets
- All databases that `databaseA` points to or references and their data sets

If a database contains no external pointers or references and is not pointed to or referenced by any other database, it alone is the data set. The application developer is often the best source for obtaining information about external pointers or references. If the developer is not available, you can use the `osaffiliate` utility to list a database's affiliation tables. (An affiliation table lists the cross-database references — for example, an affiliation table of databaseA identifies the databases to which `databaseA` points.) You can use the information in the affiliation tables to establish a database's data set. For more information, see osaffiliate on page 135.

Once you have identified the databases in the data set, if you move all the databases of the data set as a unit, no further action is usually required. For example, if there are references among the three databases

```
host:/usr/ed/db1
host:/usr/ed/dbs/db2
host:/usr/ed/dbs/db3
```

and they are moved to

```
host:/usr/fred/db1
host:/usr/fred/dbs/db2
host:/usr/fred/dbs/db3
```

or

```
hostess:/usr/ed/db1
hostess:/usr/ed/dbs/db2
hostess:/usr/ed/dbs/db3
```

no action is required so long as the affiliation tables use relative pathnames. If you move one of the set or change the name of one or more of the databases, you must use the `osaffiliate` utility to update the others to reflect the change. See osaffiliate on page 135.

Schema databases

When you are operating on schema databases, you need not identify the data set. A schema database does not have cross-database pointers and references.

Your application's executable files may have references to schema databases. Use `ossetasp` to determine whether the path name is embedded in the application and to update the path name as required. Applications can use the `set_application_schema_pathname()` function of the `objectstore` class to choose the location of the schema database at run time. If you use this function, you must work with the application developers to determine how best to keep schema database path names consistent. See also ossetasp on page 221.

Operating on a data set

After you identify a data set, you can perform administrative functions on the entire set. If you rename or move an individual database in the set, use the `osaffiliate` utility (described in osaffiliate on page 135) to fix any external pointers and references.

Moving, Renaming, and Dumping Databases

Before moving, renaming, or dumping a database, do the following:

1. Run the `osverifydb` utility (described in osverifydb on page 258) to confirm that no bad pointers or dangling references exist. This utility detects inconsistencies; it does not fix them.

   If you are moving a database within a rawfs, verification is not necessary.

2. Run the `ossize` utility (described in ossize on page 238) to ensure that cross-database references are established correctly.
3. Perform steps 1 and 2 on all databases in the data set.
4. Run the `ossvrchkpt` utility (described in ossvrchkpt on page 241) to ensure that all data has been propagated from the log to the affected database or databases.

To avoid administrative difficulties,

- Encourage developers to design database hierarchies that can be moved easily.

- Keep all databases in a set in a single directory.

Managing Server-remote Databases

Databases are usually local to the server. However, you can enable a server to control databases that reside on a remote host. See Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 265, for details.

Databases and Applications on CD-ROM

You can place an ObjectStore application on a CD-ROM and run the application from the CD-ROM. You can access read-only databases on the CD-ROM. To do this, you must allocate a server log on a local writable disk. The server itself (the executable files that make up the server) can reside on the CD-ROM.

You should validate the schemas before you place an application or database on a CD-ROM. In other words, run the application on the database in question (in update mode) and then put the application or database on the CD-ROM. This validates the schemas and sets the appropriate cache state in the databases. The result is improved performance because schema validation is not needed when the application opens the database on the CD-ROM.

For information about schema validation, see Chapter 9, Schemas, in the *C++ API User Guide*.

Managing Database Fragmentation

A database becomes fragmented when its data is allocated in noncontiguous sectors that are scattered throughout the disk. Although some degree of database fragmentation is unavoidable, severe fragmentation can impact the performance of the ObjectStore server.

Fragmentation can occur in two areas of an ObjectStore database:

- The mapping of persistent objects to offsets within an ObjectStore cluster — that is, the *logical structure* of a database
- The mapping of clusters to database files and of files to disk storage — that is, the *physical structure* of a database

The following sections describe fragmentation in the logical structure and physical structure of an ObjectStore database. These sections also include information about detecting and controlling fragmentation.

Fragmentation in the Logical Structure

The logical structure of a database can become a source of fragmentation when the client application does either of the following:

- Allocates noncontiguous storage for persistent objects that are accessed together.
- Creates and deletes many persistent objects in the same cluster.

Noncontiguous allocation occurs when the application does not cluster data to match its access patterns. For example, if the application always accesses the objects x and y together, then they should be allocated in the same cluster and as close to each other as possible. If they are allocated in different clusters or in noncontiguous areas of the cluster, the server must transfer each object separately, resulting in increased disk I/O and network activity and possibly decreased performance.

The ObjectStore API enables you to control the placement of objects in the database; see Clustering in Chapter 3, Programming with Persistent Storage of *C++ API User Guide*. For a discussion of clustering and its impact on performance, see the ObjectStore Technical White Paper, *Designing C++ Applications for Scalability and Performance*.

Creating and deleting many persistent objects can fragment a database by resulting in numerous free regions intervening between valid persistent objects in the same cluster. One way to prevent this type of fragmentation is by pool allocation and by using data structures to keep track of where the objects and free regions are in the pool. For a simple implementation of pool allocation, see Stanley B. Lippman's *C++ Primer*, 2nd edition (Reading MA, 1991), pages 149 - 150.

You can defragment a database that contains many unwanted free regions by using the `oscompact` utility, as described in oscompact on page 160.

Fragmentation in the Physical Structure

The physical structure of a database can become fragmented in either of the following circumstances:

- The mapping of the data part of a cluster onto sectors in the database file or sectors in rawfs partitions results in the entire cluster's data not being stored in a single contiguous range of sectors. This fragmentation can also happen for metadata tables but is rare because these tables are composed of a number of small units that are accessed separately.
- The mapping of a database file or rawfs partition file onto sectors of one or more disk drives results in the entire file not being stored in a single contiguous range of sectors.

In both cases, the root cause is the growth policy for allocating physical disk space (that is, sectors in a database file or in rawfs partitions) to clusters. If the policy is too small (for example, if it is biased towards minimizing consumption of disk space), a cluster's disk space will only grow by a small number of sectors. If many clusters are growing at the same time, their sectors will be interleaved and each cluster's disk space will end up fragmented into many small fragments.

Checking for Fragmentation

If application performance is unexpectedly slow and the bottleneck seems to be in the ObjectStore server rather than in the client, you should check to see if your database is fragmented. A small amount of fragmentation is usually acceptable, but if clusters have more than a few fragments or there are fragments smaller than a page (4K or 8K on most platforms), fragmentation may be slowing performance.

ObjectStore provides the following tools for detecting and reporting on database fragmentation:

- `os_database::get_fragmentation()`. This function returns statistics about fragmentation in the physical structure of a database. For more information, see `os_database::get_fragmentation()` in Chapter 2 of *C++ API Reference*.
- `ossize -f`. When invoked with the `-f` option, the `ossize` utility displays fragmentation statistics in report format. For more information, see ossize on page 238. You can also use the `ossize` utility programmatically, as described in `os_dbutil::ossize()` in Chapter 2 of *C++ API Reference*.

Minimizing Fragmentation

One way to minimize fragmentation in the physical structure of a database is to change the policies for growing clusters, database files, and expandable rawfs partitions. These policies can be changed by means of the server parameters described in the following sections of Chapter 2, Server Parameters, on page 77:

- `Cluster Growth Policy` on page 85
- `Database File Growth Policy` on page 86
- `RAWFS Partition Growth Policy` on page 96

In the following paragraphs, it is assumed that you understand the syntax for specifying these server parameters, which is also described in the above-mentioned sections.

The `Cluster Growth Policy` parameter is probably the most useful of the three. If disk space usage is your primary concern, you might want to set this parameter to a smaller value than the default. For example:

```
Cluster Growth Policy: 512
```

Alternatively, you might consider a policy that grows clusters in increments equal to the client page size — one of the following, depending on the type of client hardware you use:

```
Cluster Growth Policy: 4K
Cluster Growth Policy: 8K
Cluster Growth Policy: 16K
```

If most of your clusters are very large, you might decide to use a policy that grows large clusters by very large amounts, while still growing smaller clusters by small amounts. For example, you could append three more items to the default policy to produce this policy:

```
Cluster Growth Policy: 512 20K, 1K 40K, 2K 80K, 4K 160K, \
  8K 320K, 16K 640K, 32K 1280K, 64K 2560K, 128K 5120K, 256K
```

Your choice of cluster growth policy should be determined by how much fragmentation you want to tolerate and how much disk space you are willing to waste by allocating more space to a cluster than it will ever use.

The `Database File Growth Policy` and `RAWFS Partition Growth Policy` parameters are useful for addressing fragmentation of the mapping of a database file or rawfs partition file onto sectors of one or more disk drives. If your databases are

very large, you might consider increasing the growth increment beyond the default of 1 MB at a time in the default `Database File Growth Policy`. For example, the following policy will cause database files to grow by up to 10 MB at a time:

```
Database File Growth Policy:   4K 40K, 10% 100M, 10M
```

Adjusting the `Database File Growth Policy` and `RAWFS Partition Growth Policy` parameters can also affect cluster fragmentation by leaving more space free in the database file or RAWFS. ObjectStore can take advantage of this free space to allocate clusters with buffer zones of free space between them that allow multiple clusters to expand contiguously without bumping into each other, thus avoiding fragmentation.

Presizing

Another way to minimize fragmentation is to presize the cluster or database file. Presizing is best used when the final or desired size of the cluster or database file is well understood and the application that creates it can easily have the presizing logic added.

Rather than allow a cluster to grow incrementally as persistent objects are allocated in it, you can presize the cluster by calling the `os_cluster::set_size()` function. Cluster presizing is best done in the same transaction that creates the cluster, but it can also be done later. Either way, no other clients (except MVCC readers) can read or write to the cluster concurrently. Presizing a cluster is likely to cause much less fragmentation than gradual growth would cause. If presizing can be used to fix particular problem clusters, there may be no need to tune the `Cluster Growth Policy` server parameter.

Similarly, a database file can be presized by calling the `os_database::set_size()` or `os_database::set_size_in_sectors()` function. The presizing operation can be slow because the host operating system requires that the entire file be written at the time it is presized, and disks have a limited I/O speed, typically on the order of 1 GB per minute. Thus, database file presizing is best done when setting up a set of databases rather than during production when response time is important. If presizing can be used to fix particular problem databases, there may be no need to tune the `Database File Growth Policy` parameter.

For information about the presizing API, see the following sections of Chapter 2 in *C++ API Reference*:

- `os_cluster::set_size()`
- `os_database::set_size()`
- `os_database::size_in_sectors()`

You can also presize a cluster or database file on the command line, using the `osdbcontrol` utility, as described in osdbcontrol on page 169.

Defragmenting the Physical Structure

The best way to defragment an existing database is to back it up and restore it, as follows:

1. Use the `osdbcontrol` utility to take it offline.
2. Change its access mode or name so that clients cannot use it.
3. Bring the database back online with the `osdbcontrol` utility.
4. Back up the database with the `osbackup` utility.
5. Delete the original database with the `osrm` utility.
6. Restore the database with the `osrestore` utility.
7. Check that the fragmentation has been removed with the `ossize` utility.
8. Restore the database's original name and access mode so that clients can use it again.

If you have enough free disk space, you can use the `oscopy` utility instead of Steps 4 - 6. Use `oscopy` to copy the database to a new database with a temporary name and then use the `osmv` utility to rename the database to its original name.

For information about the ObjectStore utilities, see Chapter 4, Utilities, on page 131.

Overview of the Backup/Restore Facility

ObjectStore provides the following utilities to back up and restore databases without interrupting the databases' availability:

- `osbackup` and `osarchiv`

  These utilities back up specified databases to a secondary storage location. The `osbackup` utility performs backups according to the level that you specify; that is, it copies clusters that have been modified since a certain date. The `osarchiv` utility performs snapshots of pages modified since the previous snapshot or backup. You specify the interval between snapshots.

- `osrestore` and `osrecovr`

  These utilities copy data from backups and archive media to your disk. The `osrestore` utility primarily restores databases from sources created by the `osbackup` utility. The `osrecovr` utility primarily restores data from archive files created by `osarchiv`. You can specify a point in time to which you want to recover data.

These utilities provide protection from catastrophic data loss due to hardware failure and can be used to transport databases from one location to another. Run the utilities in the following order:

1. Run the `osbackup` utility (described in osbackup on page 143) according to a schedule that you determine.
2. Keep the `osarchiv` utility active. See osarchiv on page 137.
3. Run the `osrestore` utility (described in osrestore on page 208) to recover data backed up by the `osbackup` utility.
4. Run the `osrecovr` utility (described in osrecovr on page 199) to recover data from archive logs.

Online Backup and Archive Logging

Online backup is the first step in guarding against data loss, but it cannot help you recover modifications made after the backup was performed. Archive logging provides the information you need to recover modifications made after a backup. Online backup gives you a starting point for using archive logging.

Online backup and archive logging provide these features:

- Interdatabase transaction consistency. Databases are consistent both internally and across the entire set of databases being backed up.
- Distributed backup capability, which allows transaction-consistent backups of interdependent distributed databases. You can back up databases on different servers in the same backup operation.
- Concurrent read and write access by ObjectStore applications to databases in the backup set. That is, online backup never obtains locks on data in the backup set, so standard ObjectStore transactions can proceed normally.
- Support for full and incremental backup.
- Platform-independent format of backup archives, which allows databases to be moved from a server on one architecture to a server on another architecture. (This does not affect whether or not the clients of a given architecture can access the database. See Chapter 5, Building Applications for Multiple Platforms in *Building C++ Interface Applications*.)
- Ability to generate a single archive log for distributed databases.
- Ability to perform batch transactions to reduce the amount of data that must be archived.

The server transaction log is not affected by archive logging.

Online Restore and Recovery

The `osrestore` utility restores databases from a backup image, restoring either the entire backup set or selected databases within the set. Although `osrestore` runs with the ObjectStore server online, ObjectStore applications cannot access databases that are being restored until the entire restoration process is complete.

Online restore and recovery offer these features:

- You can almost always recover a committed transaction if archive logging was being performed for the database when the transaction was committed. Note, however, that there is some delay between the time when a transaction is committed and the time when it is written to the archive log. If the crash occurs during this interval, the transaction will not be written to the archive log.
- You can recover databases back to a transaction-consistent state at a specific time by restoring backup images and then applying committed changes from the archive log.
- Recovery time is fast because the archive log stores modified data and not the set of operations that modified the data.

- You can recover modifications from backup images and from archive logs in the same operation.

See Backup Strategies on page 50 for particular suggestions about using ObjectStore's backup and restore tools in a manner that best suits your environment.

Backup Strategies

When you develop a backup strategy, you need to consider certain conditions of your database environment. The questions to think about include the following:

- How large are your ObjectStore databases?
- How far back do you need to back up?
- How much downtime to perform backups can you tolerate, if any?
- How much time can you tolerate for recovery?

The following sections provide suggestions and examples of methods and tools you might use, depending on your answers to the previous questions.

General Backup Practices

ObjectStore Technical Support recommends, at a minimum, a nightly backup of ObjectStore databases. The `osbackup` command is an online utility that runs as if it were an ObjectStore multiversion concurrency control (MVCC) client. It does not cause any concurrency conflicts.

With `osbackup`, you can specify a backup level from 0 to 9. Files that have been modified since the last backup at a lower level are copied to the backup image. For example, suppose that you did a level 2 backup on Monday, followed by a level 4 backup on Tuesday. A subsequent level 3 backup on Wednesday would contain all files modified or added since the level 2 (Monday) backup.

You can store the backup image on tape or on disk. If you are backing up to disk, use a separate disk for the backup images. This allows reads to occur on the source disk and writes on the target disk, which produces a faster backup. This also ensures that a fatal hardware failure of the database disk does not destroy the backup images.

Windows

You might choose to run Microsoft's Schedule service or use the `at` command to invoke this operation automatically each night.

UNIX

You can take advantage of the `cron` facility on UNIX systems to automatically invoke the backup operation each night.

After the backup is complete, the backup images can then be moved to tape and eventually be stored in a safe location. For determining the best backup strategy, it is important that the sizes of production databases be estimated. If the databases are not in the gigabyte range, it is better to perform a level 0 (full backup) every night, as opposed to doing additional incremental backups. The advantage is that, in the

event of a failure, only a single backup image needs to be restored. The following is a sample backup strategy:

Sun.:	Backup level 0
Wed.:	Backup level 1
Mon., Tues., Thurs., Fri., Sat.:	Backup level 2

For example, if you need to restore the databases to their state on Saturday morning, you must restore the backup images from Sunday night, Wednesday night, and Friday night.

osbackup command flags

Following is a sample backup command:

```
osbackup -i bkup_rec -l 0 -f full_bkup.img sample1.db \
    sample2.db sample3.db
```

- The `-i` flag specifies the *incremental record file* — a file that contains information about the databases that have been backed up and when they were backed up. The `osbackup` utility uses this information to determine those clusters and segments within a database that have been modified since the last backup at a lower level. The utility then backs up only modified clusters. The incremental record file is comparable to the archive record file for `osarchiv`.

  Use the same incremental record file for both `osbackup` and `osarchiv`. If you do not, `osarchiv` starts its archiving operation with a full copy first. Make sure that you use the most recent backup and archive images.

- The `-l` flag specifies the level of the backup. You can specify an integer from 0 to 9. Files that have been modified since the last backup at a lower level are copied to the backup image. Backup is incremental at the cluster level, meaning that a cluster is only backed up if it has been modified since the last backup at a lower level. A level 0 backup (the default) backs up all clusters in all specified databases.
- The `-f` flag specifies the location of the backup image.

See osbackup on page 143 for further explanation of the options for this utility.

Archive Logging

The `osarchiv` utility takes much smaller pictures of the database than `osbackup` takes. ObjectStore Technical Support recommends that you always run the `osarchiv` utility, even during an `osbackup`. The advantage of this is that in the event of a system failure during the `osbackup` process, no data is lost. The `osarchiv` utility performs snapshots of pages modified since the previous snapshot or backup. You specify the interval between snapshots. The `osarchiv` utility records all transaction activity for specified databases.

Because the `osarchiv` process requires a backup image as initial input, the script responsible for this operation might choose to stop `osarchiv` and restart it each night after the `osbackup` process is completed successfully. Avoid updates in the window of time following the start of `osbackup` and the startup of `osarchiv` so that `osarchiv` does not have to dump full clusters. You can move the archive file from disk to tape at this time if disk space is an issue.

`osarchiv` switches to the next archive file in the sequence when it encounters the maximum size allowed for an archive file. You can specify the maximum amount with the `-s` flag. The default maximum size is 2 MB. If `osarchiv` runs out of disk space for archive files, it notifies you and suspends activity until additional disk space is available.

The utility uses the following naming convention for archive files:

`YYYYMMDDHH.ext`, where

Variable	*Meaning*
`YYYY`	Year
`MM`	Month
`DD`	Day
`HH`	Hour
`ext`	Extension of the form `aaa`, `aab`, `aac`...

If you decrease the time between snapshots, you also decrease the number of transactions recorded in each snapshot. Using shorter intervals between snapshots has the effect of keeping the archive more up to date and keeping the amount of data that needs to be archived smaller at each interval. However, because each snapshot causes information to be written to the archive file, even if no data modifications are being recorded, frequent snapshots can consume space in the archive file unnecessarily. Longer intervals can reduce the amount of data being logged in cases in which the same data is modified by multiple transactions. In such cases, only the most recent copy of the committed data needs to be logged.

osarchiv command flags

The following is an example of an `osarchiv` command:

```
osarchiv -a bkup_rec -d archive_img -i 30 sample1.db \
    sample2.db sample3.db
```

Remember that the incremental record file created with `osbackup` is the starting point for `osarchiv`. In the previous command-line example:

- The `-a` flag specifies the path of the *archive record file* that `osarchiv` uses to record the segment change IDs for the archive set. The `osarchiv` utility updates this file each time it successfully records committed changes to the archive set. This activity is known as *taking a snapshot*. The `osarchiv` archive record file is equivalent to the incremental record file for `osbackup`. It is usually the same file.
- The `-d` flag specifies the directory in which to create the archive log files. You cannot perform archive logging directly to a tape device.
- The `-i` flag specifies an integer that `osarchiv` uses as the interval between snapshots. By default, this interval is in seconds, but you can append `m`, `h`, or `d` to the interval value to indicate minutes, hours, or days. For example, `-i 60` and `-i 1m` both specify an interval of one minute. When the interval is not `0`, `osarchiv` takes a snapshot immediately after being initiated and then every interval *n* seconds (or minutes, hours, or days) thereafter. When you do not specify an interval, it defaults to `0`, which means that snapshots are not taken automatically.

See osarchiv on page 137 for further information about the options for this utility.

Special Considerations for Large Databases

The `osbackup` procedure backs up approximately 9 GB an hour. The backup can take much longer under certain load conditions, such as when updates are occurring on large numbers of pages while the backup is running. Contrast this time requirement with an operating system backup that averages faster backups per hour. If you have very large databases to back up, the advantage of an online back up with `osbackup` might be less important than the amount of time it takes to do the backup. You might benefit more from shutting the ObjectStore system down and backing up the file system by using operating system commands, as described next.

The most dependable and least time-consuming method of backing up your data is to depend on the operating system backup and the ObjectStore archiver. The sequence of activities you need to complete is summarized in the following list:

1. Make sure `osarchiv` is running.
2. Run `osdbcontrol -offline` for the database.
3. Shut down `osarchiv`.
4. Run the file system backup.
5. Run `osdbcontrol -online` for the database.
6. Restart `osarchiv`.

The next section describes the way `osrestore` and `osrecovr` work.

Restore and Recovery Options

You should use the `osrestore` utility to restore databases from the most recent backup images created previously by `osbackup`. Recover any archive images created by the `osarchiv` utility by using the `osrecovr` utility.

Using both osrestore and osrecovr

When you are restoring databases to a specific point in time, make sure you run `osrestore` before you run `osrecovr`. The following `osrestore` command provides an example that assumes a full backup was performed:

```
osrestore -f full_bkup.img
```

- The `-f` flag specifies the location of the backup image. The `osrestore` utility prompts you for incremental backup images you might want to apply after this (you use this option if you have incremental backup images resulting from an incremental backup procedure).

  After the database has been restored, you can restore your database to a point in time by running the `osrecovr` utility with a command such as

  ```
  osrecovr -F archive_list.txt
  ```

- The `-F` flag specifies the location of a file containing the names of all the archive images. For example, `archive_list.txt` might contain

  ```
  C:\name\user>type archive_list.txt
  archive_img\1999020719.aaa
  archive_img\1999020719.aab
  ```

```
archive_img\1999020720.aac
archive_img\1999020720.aad
archive_img\1999020720.aae
archive_img\1999020720.aaf
```

To recover to a specific date and time, use a command such as the following:

```
-F archive_list.txt -D 2/8/1999 -r 19:59:45
```

See osrestore on page 208 and osrecovr on page 199 for further explanation of options for these utilities.

Using system backup and osrecovr

Archive image files generated in a directory (specified by the `osarchiv -d` flag) can be deleted or moved to tape at your discretion. In the event of a failure, you can first restore your UNIX or Windows backup files and then invoke `osrecovr` with the file name that contains a list of all the archived images in the directory specified by the `-d` flag.

The Dump/Load Subsystem

The dump/load subsystem provides a facility that allows ObjectStore users to dump databases into, and load databases from, a nondatabase format.

The ObjectStore dump/load facility allows you to

- Dump the contents of a database or group of databases to an ASCII file.
- Generate a loader executable capable of creating, given the ASCII file as input, an equivalent database or group of databases.

The dumped ASCII files from an ObjectStore C++ database are in a compact, human-readable format. You can read the files with an editor or string processing program such as perl, awk, or sed. The ASCII file is the input for the loader.

See osdump on page 173 and osload on page 192 for more detailed information about using the dump/load facility with C++ databases. For advanced topics related to customization, see Chapter 6, Dump/Load Facility, in the *Advanced C++ API User Guide*.

See osjidump on page 186 and osjiload on page 188 for more detailed information about using the OSJI version of the dump/load facility with Java databases.

Upgrading a C++ Database

This section describes how to use `osdump` and `osload` to upgrade an ObjectStore database with a Release 5.1 format to one with a Release 6.1 format.

Note

To dump an ObjectStore C++ Release 5.1 database for upgrading to Release 6.1, you must use the latest version of the Release 5.1 `osdump` utility; contact Technical Support for details on how to obtain this version.

1. Verify the existing R5.1 database using the R5.1 version of `osverifydb`. If your databases have any illegal pointers or other problems, the dump and load

operations will not work properly. Contact ObjectStore Technical Support if you osverifydb reports errors you cannot resolve.

2 Dump the databases using the latest Release 5.1 version of `osdump` to create the database table file and the ASCII representations of the databases.

This step creates the database table file and the ASCII representations of the databases. The output consists of the following files: `db_table.dmp`, *`filename`*`.scm`, and a series of files with the form *`databasename_segmentN.dmp`* and *`databasename_segmentN.fix`*.

3 Generate a Release 6.1 application schema that corresponds to the schema of the Release 5.1 database.

You will need the C++ header files and the schema source file (`schema.cc` in the example, below) that marks all persistent types that were used to create the original database. You need to change or remove any types that are not supported in Release 6.1.

On Windows, generate the schema as follows:

```
ossg -asof schema.obj -asdb schema.adb /nologo /GX /MD \
/DWIN32 schema.cc
```

On UNIX, generate the schema as follows:

```
ossg -assf dschema.cc -asdb schema.adb \
-I$OS_ROOTDIR/include schema.cc
```

4 Generate the source files for the loader application from the application schema created in step 3 by using the Release 6.1 version of `osload` with the `-emit` option. For example:

```
$OS_ROOTDIR/bin/osload -emit schema.adb
```

This will generate the source code for a loader application and the makefile you can use to build the application.

Note that the `osload` utility used in this step is the generic version supplied by the ObjectStore installation. In the next steps you will build and run your own version of `osload` that is tailored to your specific database files.

5 Include your application's header files in the generated `ldrhdr.h` file

6 Update the generated makefile as follows:

- Add your own libraries to the `USER_LOAD_LIBRARIES` variable.
- Add a variable named `USER_INCLUDES` and set it to the directory where your application's `.h` files are located.
- Add a variable named `USER_LIBS` and set it to the directory containing your own libraries
- Add `$(USER_INCLUDES)` to the end of the CCFLAGS line.
- Add `-L$(USER_LIBS)` to the end of the LDFLAGS line.

7 If necessary, customize the generated loader code.

8 Build your database-specific version of the `osload` executable from the loader source files. On Windows, use `nmake` and `makefile.w32`. On UNIX, use the `make` utility and `makefile.unx`. These makefiles are generated in step 4.

You must build the loader using ObjectStore Release 6.1 libraries.

You should build the loader in a directory different from the directory where the dumped database files (step 2) are located.

9 Generate the Release 6.1 format database by running `osload` executable you created in step 8.

```
osload -cwd -dir dirname
```

The `-cwd` option tells `osload` to generate the databases in the current working directory. The `-dir` option specifies the location of the dumped database files.

See osload on page 192 for more information on `osload`.

10 Verify the new database using the R6.1 version of `osverifydb`.

Upgrading a Java Database

This section describes how to use the `osjidump` (or `osdump`) and `osjiload` utilities to upgrade a database from ObjectStore Java interface Releases 3.0, 6.0, 6.1, or 6.2 to Release 6.3.

Note

The `osjidump` and `osjiload` utilities are only installed when you install the ObjectStore Java interface. The utilities are located in the `OSJI/bin` directory. Before dumping a database, you should obtain the latest version of `osdump` or `osjidump` for the release associated with that database from ObjectStore Technical Support.

The `osjiload` utility requires the `CLASSPATH` environment variable include the OSJI classes `OSJI/osji.jar` and `OSJI/tools.jar` as well as the postprocessed class files used in your database schema.

To upgrade your ObjectStore Java interface database:

1 Dump the database using `osdump` or `osjidump`. For dumping ObjectStore Release 3.0 or 5.1 databases, use `osdump`. For dumping Release 6.0, 6.1 or 6.2 databases, use `osjidump`.

`osdump` *`database_pathname`* or `osjidump` *`database_pathname`*

The *`database_pathname`* argument specifies the location of the database. you are upgrading.

This creates the database table file (`db_table.dmp`), files representing the database schema (*`database`*`_name.scm`), and a series of files with the form *`databasename_segmentN.dmp`* and *`databasename_segmentN.fix`* that are representations of the database with the information in ASCII . The files are created in the current working directory.

2 Load the database by running the `osjiload` utility from the directory where you want the Release 6.3 version of the database to be located.

```
osjiload -cwd -dir dirname
```

The `-cwd` option tells `osjiload` to load the databases in the current working directory. The `-dir` option specifies the location (*`dirname`*) of the dumped database files.

For information on using `osjiload`, see osjiload on page 188.

For more information on upgrading databases, refer to the *ObjectStore Migration Guide*, available at www.objectstore.com/documentation/objectstore.

Managing Users

There are two types of ObjectStore users:

- Those who run ObjectStore applications
- Those who develop ObjectStore applications

This section focuses on the environment an administrator must provide for users who run ObjectStore applications.

Running applications

To run, an ObjectStore application needs

- Client, server, and cache manager executables.
- An application schema database.
- One or more databases for storage of application data. The data can already exist or can be created by the application.

OS_ROOTDIR

References to ObjectStore executables, libraries, and utilities require a definition for the environment variable, `OS_ROOTDIR`. The value of `OS_ROOTDIR` indicates the top-level directory in the file system hierarchy containing the ObjectStore release. It serves as the prefix of various directory names used in search paths.

You can set `OS_ROOTDIR` on each host that runs an ObjectStore server or application. Alternatively, you can set `OS_ROOTDIR` in each user's environment.

`OS_ROOTDIR` allows you to change the location of the ObjectStore installation and allows users to switch to a different version of ObjectStore.

Application schema database

Every ObjectStore application has one application schema database that the schema generator produces when the application is built. The application schema database contains the definitions for all classes the application uses in a persistent context. An ObjectStore executable embeds the path name of the application schema database. Therefore, as administrator, you must ensure that the application schema database is available to the executable.

You can use the `ossetasp` utility to change the path name of an application schema database for a particular executable.

Alternatively, an application can specify its application schema database path name when it runs by calling `objectstore::set_application_schema_path()`.

Sharing schemas

Multiple executables can share an application schema database. Developers create the application schema database as part of the procedure for building the

application. The purpose of the application schema database is to ensure that the schema for an accessed database matches exactly the schema for an executable. ObjectStore compares the application schema to the database schema during execution.

Developing applications

To develop an ObjectStore C++ application, a developer needs to be able to run `ossg`, the ObjectStore schema generator. Developers use the schema generator to create the application schema database. See *Building C++ Interface Applications*.

Modifying Network Port Settings

Normally, the default settings for ports for network services are sufficient. However, you can modify them if you need to. You might want to do this if you are running two releases of ObjectStore on a machine at the same time. You might also want to modify port numbers if your site uses a firewall to prevent unauthorized network access. You can choose to have the firewall protect the default port numbers shown in Defaults for Port Settings on page 59, or you can select different port numbers that are protected.

Communication Methods

To communicate with a network daemon such as the ObjectStore server, a client requires a host address to identify the machine and a port to identify a particular process. ObjectStore finds host addresses by looking up the host name in a host table or similar facility. ObjectStore uses default ports, which can be overridden if necessary, as described in the following paragraphs. On each platform, ObjectStore supports at least two ways for processes to communicate:

- The local transport layer allows communication between processes on the same host.
- The remote network allows the local host to communicate with remote hosts. In Release 5.1 and above, this protocol is always TCP/IP.

The following table lists the communication channels for each platform:

Platform	*Local Transport Layer*	*Remote Network*
Solaris 2	`TLI_LOCAL`	TCP/IP
Other UNIX platforms	`sockets`	TCP/IP
Windows	`NSharedMemory`	TCP/IP

UNIX

For the local network, ObjectStore uses sockets or TCP/IP streams to communicate among server, client, and cache manager processes using predefined ports that you can modify if necessary. A port identifies a particular application on that machine. (An address identifies a particular machine.)

Windows

For the local network, ObjectStore uses a proprietary local network based on Named Shared Memory. Server, client, and cache manager processes use an identifier instead of port addresses. You can modify this identifier in the ports file.

Defaults for Port Settings

For `NSharedMemory`, the port setting modifies the default port, so you can use the same setting for all services. The defaults are

```
connect.server_client
accept.server_client
connrel.server_client
connreg.server_client
exception.server_client
```

On the other networks, a port setting replaces the existing port setting, so you must provide unique settings for each port that you modify.

The default UNIX port settings are in the following tables:.

Process	*IP Port Number*	*TLI_LOCAL Path Name*	*UNIX Path Name*
Server to R6 cache manager	`51031`	`os_callback_v6`	`/tmp/ostore/os_callback_v6`
R6 client to R6 cache manager	`51041`	`osccom6`	`/tmp/ostore/osccom6`
Client to server	`51025`	`objectstore_server_comm`	`/tmp/ostore/objectstore_server_comm`
Notification	`51050`	`objectstore_notification`	`/tmp/ostore/objectstore_notification`

Modifying Port Settings

To modify settings, you change entries in the appropriate file shown in the following table. After you modify the ports file, you must shut down and restart the server and cache manager for the changes to take effect. When the daemons start, they detect the existence of the ports file and use the settings in it.

On UNIX and Windows, you can also set the variable `OS_PORT_FILE` to the name of a file you create.

Platform	*Ports File*
UNIX	`$OS_ROOTDIR/etc/ports`
Windows	`%OS_ROOTDIR%\ETC\PORTS`

Settings Format

Each line in the ports file specifies the port for a network service. The syntax of a line in the ports file is

net:*service*:*version*:*port*

net specifies the network type. It must be one of the values that appears in the `Network` column in the next table. Possible values vary according to the platform.

service specifies one of the following network services:

- `cache manager client` is the service the client uses to find the cache manager. This is only meaningful when *net* is a local network. (The port the cache manager listens on for clients is always accessed by local clients.)
- `cache manager server` is the service the server uses to find a cache manager.
- `cache manager notification` is the service the client uses to find a cache manager for notification receipt.
- `server client` is the service a client uses to find the server.

version specifies the software version for the cache manager client or the cache manager server. For Releases 4 and 5, specify 4. For ObjectStore Release 6, specify 6 as the version number in all cases except `server client`. This service requires a 1.

port specifies a port identifier as described in the following table:

Network	*Port Identifier*	*Port Identifier Example*
`IP` (TCP/IP)	TCP/IP 16-bit port number. See Defaults for Port Settings on page 59.	`51025`
`NSharedMemory`	The string appended to shared object name. Omitted by default.	`accept.server_client_`*`myserver`*
`TLI_LOCAL`	Path name in TLI Local namespace. No default.	`objectstore_server_com`
`UNIX`	Path name in file system. No default.	`/tmp/ostore/objectstore_server_com`

Here are some examples of port settings:

```
TCP/IP:server client:1:54432
NSharedMemory:server client:1:new_server
UNIX:server client:1:/tmp/.ostore/objectstore_server_comm
UNIX:cache manager client:6:/tmp/.ostore/objectstore_client_cache
UNIX:cache manager server:6:/tmp/.ostore/objectstore_server_cache
```

If a ports file is not present, ObjectStore uses default settings.

Running Two Servers on One Host

You can run two ObjectStore servers on the same host under some circumstances. To do so, you must specify different ports in the ports file. This lets clients know which server to contact.

Same Release (6.1 Or Later Only)

Under release 6.1 or later, you can run more than one ObjectStore server on the same host so as to allow a client to address the servers at the same time. The servers must be of release 6.1 or later. This configuration is useful when enabling failover. For more information, see Configuring More Than One Logical Server on page 293.

Different Releases

You can also run two servers of different releases on the same host — for example, a Release 5 server and a Release 6 server. To do so, you need to change only the server client ports for one of the servers. The cache manager ports are already different in different major versions of ObjectStore.

Before Release 6.1, all servers that a client can communicate with must have the same network port number. This means that if you run two servers on one host, any single client can access one of those servers but not the other. It also means that if a client accesses servers on more than one host, you must change the server port settings on all affected hosts.

UNIX notes

On System V Release 4 systems, such as Solaris 2, specify a combination of `TLI_LOCAL` and `TCP/IP` statements. Note that the `TLI_LOCAL` domain port names exist in a separate, flat namespace.

Windows

On Windows systems, you can specify statements for `NSharedMemory` and TCP/IP.

When Your Application Uses Notification

When an ObjectStore application uses notifications, the client automatically establishes a second network connection to the cache manager on the local host. The application uses this connection to receive (and acknowledge receipt of) incoming notifications from the cache manager. (Outgoing notifications are sent to the server, not to the cache manager.)

This all happens automatically and transparently, so there is no relevant API. However, as with all network connections, you might want to control the network port that is used by the connection. You can use the ports file in the usual way to control the use of network ports. The following information should appear in the networks ports file:

- Name of service - specify `cache manager notification`
- Default IP port number - specify `51050`
- Default UNIX local socket file name - specify `/tmp/ostore/objectstore_notification`

How a Client Locates the Server for a

Database

How does a client determine the server to communicate with? The answer depends on whether the database being created or opened is a file database or a rawfs database.

When Accessing a File Database

When a client tries to open or create a file database, the client has a path name for the database. If the pathname contains an explicit "`host:`", the client uses the server on the specified host. Otherwise, the client looks for the server on the machine that has the disk on which the database is or will be stored. Normally, this disk is local to the machine with the server. In other words, the database is *server local*.

If the client cannot find a server that is local to the disk, it signals an error unless there is a locator file.

A locator file allows a client to access *server-remote* databases. Server-remote databases are stored on disks that are not local to an ObjectStore server. See Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 265.

If there is a locator file, the client checks it to determine the server to communicate with.

When Accessing a Rawfs Database

When a client tries to open or create a rawfs database, the rawfs host is specified in the path name of the rawfs database. See Rawfs Databases on page 19.

Managing ObjectStore on Multiple Platforms

When you manage a site that runs ObjectStore on a variety of platforms, there are issues to consider that do not exist when you run ObjectStore on a single platform.

Using Multiple File Systems

ObjectStore is not dependent on a particular file system. If the usual file system calls (such as open, read, write, and close) work on the file, ObjectStore can use them successfully.

Translating Path Names

You might run an ObjectStore application on both PCs and UNIX systems. UNIX uses forward slashes (/) in path names; PCs use backslashes (\). How does ObjectStore handle this?

Suppose you have a UNIX server along with some Windows clients.

If you are using something like an NFS client to mount the UNIX file system, you use the client's native syntax. For example, on a PC on which drive Q is NFS-mounted

on `husky:/desktop1`, ObjectStore expands the file name `Q:\jet\my.db` into a reference to `/desktop1/jet/my.db` on the server host `husky`.

If you do not use file mounting, you can use server-relative path names of the form `husky:/desktop/jet/my.db`. ObjectStore interprets the part of the path name that follows the colon (:) according to the syntax of the server host (UNIX in this example).

ObjectStore supports multibyte encoded path names. This support extends to environments where client and server machines use different multibyte encodings.

Ownership and Locks

ObjectStore uses a patented system of *ownership* and *locks* for managing concurrent access to data.

Read or write ownership

A client has ownership of a page when the page is in the client's cache. Ownership is managed automatically by ObjectStore as pages move in and out of the cache and is automatically acquired when an application attempts to access persistent data.

Read or write locks

A client has a lock on a page when the client's current transaction is using persistent data on the page. Like ownership, locks are managed automatically by ObjectStore and are automatically acquired when an application attempts to access persistent data. The application can also explicitly acquire a lock by calling the `objectstore::acquire_lock()` function. Once acquired, all locks are held until the end of the transaction and released at the end of the transaction. For more information about explicitly acquiring locks, see `objectstore::acquire_lock()` in Chapter 2 of *C++ API Reference*.

To access persistent data, the client must have both ownership and a lock on the page containing the data. Ownership provides the client with a copy of the data that can be mapped into the client's virtual memory. The lock ensures that clients concurrently accessing the same data do not interfere with each other. In other words, the lock ensures consistent results, as if only one transaction were executed at a time.

Ownership and locks are acquired either for reading or for writing. To read persistent data, the client must have read or write ownership and a read or a write lock. To modify persistent data, the client must have write ownership and a write lock.

If a client has a read lock on a page, no other client can change that page until the client releases the lock by committing or aborting the transaction. If a client has a write lock on a page, no other client can read or write that page until the client releases the lock by committing or aborting the transaction.

How Ownership and Locks are Related

In order to acquire read or write ownership of a page that is not already owned, a client must have a read or write lock on the page. If the client has ownership of a page and wants to acquire a lock on the page, it can acquire the lock without communicating with the server because no conflict with other clients is possible. Otherwise, it must send a request to the server and wait for the server to grant the lock. Thus, if a client acquires a lock in one transaction and needs the same lock in the next transaction, the client can acquire the lock quickly (unless the page has been called back) because it still has the ownership.

A client can own a page without having a lock on it. After the completion of the transaction in which the lock was acquired, the client releases the lock but retains ownership. A client can also own a page without locking it when the page is prefetched into the cache.

A client can also have a lock on a page without owning it. This situation can occur when the client has ownership of a page and has the page locked. If the client's cache is full, it must evict some pages, thus releasing ownership but without releasing the locks on those pages. A client can also have a page locked without owning it if the client acquires a lock without fetching a page into its cache; in other words, it has the lock but not ownership.

Ownership can be retained beyond the end of a transaction; locks cannot. Locks can still be held on pages after they have been evicted from the cache; ownership cannot.

Lock Management

The server is the lock manager; it takes care of coordinating locks and ownership among multiple clients. The server makes sure that the lock acquired by a client is always valid with respect to locks held by other clients, even when a client acquires the lock without communicating with the server. If a client requests a page that is already owned by another client, the server sends a callback message to the owning client before letting a conflicting lock be acquired, as described in Ownership Conflicts on page 65.

A server grants a read lock to as many clients as request it. Many clients can have read locks on a page at the same time, but only one client at a time can have a write lock. When a client has a write lock, no clients can have a read lock.

If a client requests read ownership or a read lock, and another client has a write lock on the requested page, the first client must wait until the second client ends its transaction and releases the write lock.

If a client requests write ownership or a write lock, and other clients have read or write locks on the requested page, the first client must wait until all the other clients end their transactions and release their locks.

A server grants read ownership to as many clients as request it. Many clients can have read ownership at the same time, but only one client at a time can have write ownership. When a client has write ownership, no other clients can have read ownership.

If a client requests read ownership or a read lock, and another client has write ownership of the requested page, the first client must wait until the second client releases write ownership.

If a client requests write ownership or a write lock, and other clients have read or write ownership of the requested page, the first client must wait until all the other clients release their ownership.

Ownership Conflicts

When an ownership conflict occurs, the server sends a *callback message* to the cache manager on the client's host requesting the client's ownership to be released. The cache manager determines whether ownership can be released or the conflicting client must wait.

Clients release ownership in the following situations:

- A callback message finds that the page is not locked.
- A callback message that requests release of write ownership finds that the page is only read-locked, not write-locked. In this case, ownership is downgraded from write to read.
- The transaction ends, releasing a lock, but a callback message was received before the lock was released.
- The client closes the database.
- The client terminates its connection with the server.

Locks and Nested Transactions

Aborting a nested transaction releases locks acquired during the nested transaction but does not release locks acquired in the parent transaction. Committing a nested transaction does not release locks; the locks become the property of the parent transaction.

Locks, Ownership, and MVCC

Multiple Version Concurrency Control (MVCC) modifies the normal handling of ownership and locks. When a client has a database open for MVCC, its read locks and read ownership of pages in the database never cause other clients to wait. Instead, the MVCC client just sees an older, but consistent, snapshot of the database while other clients work with the newest snapshot. Callback messages are still used, but only to facilitate bringing the MVCC client's snapshot up to date when it starts its next transaction. An MVCC client cannot have a write lock or write ownership.

API for Lock Management

Typically, ObjectStore performs lock management. There are, however, API features that allow a client to control whether a client waits for a lock. For more information about ownership and locks, see Chapter 2, Advanced Transactions, in the *Advanced C++ API User Guide*.

ClearCase Virtual File System (MVFS)

If your ObjectStore applications reside in a ClearCase virtual file system (MVFS), they might have trouble locating an ObjectStore server. To overcome this problem, you should set up locator files or use ObjectStore/Single. See Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 265, for more information on using locator files. See Chapter 6, Working with ObjectStore/Single, in *Building C++ Interface Applications* for more information about ObjectStore/Single.

Troubleshooting

This section provides examples of problems you might encounter. In general, when you receive an error message you should

- Try to define what the problem is
- Determine the ObjectStore resource that is affected
- Obtain more information using the procedures indicated in the following sections

These steps allow you either to take a corrective action that resolves the problem or to provide Technical Support with enough information to resolve it.

Where to begin?

If you have a problem with ObjectStore, the problem might lie in your client application, the server, or the cache manager. It might also be due to problems with your database or databases or application schema database or databases, or it might be due to operating system or network configuration problems. To further define the problem, try these procedures:

- To isolate the problem, review the Installation Guide for your platform. Ensure that you are running a supported version of the operating system and have installed all required patches.
- If the problem is that the server or cache manager is crashing or not responding, see Debugging the ObjectStore Server on page 67 or Debugging the Cache Manager on page 72.
- If you suspect problems with a database, use `osverifydb` to verify the content of the database. It is good practice to run `osverifydb` occasionally to confirm that your database is not corrupt. By maintaining a history of `osverifydb` results, you can identify when corruption appears.
- If the problem is with your client application, see Environment Variables for Debugging Client Applications on page 74.

The Product Support department maintains an extensive archive of answers to frequently asked questions (FAQs) at the ObjectStore Technical Support Web site (`support.objecstore.net`). Search for keywords pertaining to your problem.

Debugging the ObjectStore Server

When the server starts, exits, or encounters a severe error, it records that information in its text output file. Examine this file for error messages. The files to examine are

- UNIX: `/tmp/oss_out`
- Windows: `%OS_TMPDIR%\OSSERVER.TXT`

Note

These are default locations, and they may have changed during the installation. Check the correct file locations for your installation.

Ensure that you are starting the server with the correct permissions. For example, on UNIX you must be `root`.

Consider what the server needs to do when it starts up. See What Does the Server Do? on page 20.

If the server fails to start, you can try starting it by hand with additional arguments to enable debugging output. On UNIX, as `root`, enter

```
osserver -F -d 5
```

On Windows, in a console window, enter

```
osserver -con -d 5
```

If you want to debug a server that is managed by Sun Clusters, see Debugging an ObjectStore Server on Sun Clusters on page 71 for information about starting up the server.

The server sends debug printouts to standard output, describing its actions. The number after `-d` is the debug level. The levels available are described in the following paragraphs.

You can confirm that the server has started successfully by using the `ossvrstat` command.

If the server starts successfully and you want to turn on debugging output just before a test run, you can use the `ossvrdebug` utility with a command such as

```
ossvrdebug server_host_name debug_level
```

This command turns on debug output from the named server, and `ossvrdebug 0` turns it off. The higher the debug level, the more verbose the output and the more server performance is affected. The levels are

Level	*Description*
0	Turns `ossvrdebug` off.
1	Reports client connect or disconnect.
2	Reports messages received.
3-8	Provides expanded message information.
9	Provides full message information.

The following sections discuss three different error messages you might receive when trying to start the server:

Missing Transaction Log File

Error message

```
021007 114942.812 FAILURE in ObjectStore Release 6.1 Alpha 1
Database Server
Compiled by test1 on Sep 1 2002 at 08:54:01 in janus_bl1.nt
The log file "D:\OSSERVER.LOG" does not exist.
You need to create a new log file. A new log can be created using
the -ReallocateLog switch to osserver. This switch will make
osserver create a new log and then exit.
```

Discussion of message

The resource affected is the server. The problem is that the server cannot find the transaction log file. Do not use the `-ReallocateLog` option until you have eliminated all possibility of restoring the transaction log. Using that option risks corrupting databases that had unpropagated updates at the time the log was lost, as described in Initializing the Transaction Log on page 219.

To obtain more information, you can do the following:

- Check the content of the server output file. For the name of the file on your platform, see the beginning of this section, Debugging the ObjectStore Server on page 67.
- If there is a parameters file, check its content for the name of the log file.
- If the log is not in the rawfs, confirm its existence in the location specified in the parameters file.
- Try to start the server in debug mode.

It is possible that someone accidentally deleted the log or that the log might not have the correct permissions.

Invalid Transaction Log File

Error message

```
021007 115104.781 FAILURE in ObjectStore Release 6.1 Alpha 1
Database Server
Compiled by test1 on Sep 1 2002 at 08:54:01 in janus_bl1.nt
The log file "D:\OSSERVER.LOG" does not look like a log file.
This may be because the log file is from a previous release of
ObjectStore or because it has never been initialized as a log file.
You need to create a new log file. A new log can be created
using the -ReallocateLog switch to osserver. This switch will
make osserver create a new log and then exit.
```

Discussion of message

The resource affected is the server. The problem is that the server's transaction log file exists but does not have the correct contents for a valid transaction log file. It is possible that someone accidentally replaced the log file with a non-log file or that the log file was corrupted.

Do not use the `-ReallocateLog` option until you have eliminated all possibility of restoring the correct transaction log. Using that option risks corrupting databases that had unpropagated updates at the time the log was damaged, as described in Initializing the Transaction Log on page 219.

Corrupted Transaction Log File

Error message

```
021007 120115.750 FAILURE in ObjectStore Release 6.1 Alpha 1
Database Server
Compiled by test1 on Sep 1 2002 at 08:54:01 in janus_bl1.nt
<maint-0020-0201>Unhandled exception in Main Thread:
<maint-0019-0005>transaction log contains invalid data
<maint-0019-0909>invalid log record type (158)
(err_satmgr_invalid_log)
```

Discussion of message

The resource affected is the server. The problem is that the server detected an internal inconsistency while reading the transaction log file. It is possible that there is a bug in the instance of the server that wrote the log or in the instance of the server that is reading the log. It is also possible that you switched from a newer release of the server to an older release without first propagating the log; in this case, you can fix the error by switching back to the newer release and starting the server with the checkpoint option (`osserver -c`). It is also possible that a media failure corrupted the file.

You should save the transaction log and all affected databases and contact Technical Support for assistance.

Do not use the `-ReallocateLog` option until you have eliminated all possibility of repairing the transaction log. Using that option risks corrupting databases that had unpropagated updates at the time the log was damaged, as described in Initializing the Transaction Log on page 219. In some cases, you can delete one database and successfully start the server to propagate information about other databases, so you only have one database to recover from backup.

Sample Trace File Reading

While running a client application, you might receive a message similar to the following example of an `ossvrdebug 6` trace file:

Trace file

```
Mon Oct 07 15:38:32 2002 DEBUG 0350 [03] client #35
(1.17.1364.3453103071.1875883738.IP@192.168.105.4,
  51025/mo.exe/mo.exe/V6.0)
:
server_actions CLIENT MESSAGE 8 RECEIVED
Mon Oct 07 15:38:32 2002 DEBUG 0351 [04] client #35
(1.17.1364.3453103071.1875883738.IP@192.168.105.4,
  51025/mo.exe/mo.exe/V6.0)
:
server_actions set_lock_param(able to abort,
  wait forever for lock)
Mon Oct 07 15:38:32 2002 DEBUG 0352 [05] client #35
(1.17.1364.3453103071.1875883738.IP@192.168.105.4,
  51025/mo.exe/mo.exe/V6.0)
:
server_actions upgrade_ownership write_locking(DB handle 184,
segment #4, cluster #2, 0, 8, nl=-1)
```

Comment on trace file

As indicated by the trace file, client #35 has a read lock on the page and wants to upgrade that lock to a write lock. As shown in the next trace file, clients #32 and #34 own the page for reading, and their ownership conflicts with client #35's lock request.

Trace file

```
Mon Oct 07 15:38:32 2002 DEBUG 0356 [05] Callback Thread
  (tucker) client #32
(1.17.1516.3453103059.1875883738.IP@192.168.105.4,
  51025/mo.exe/mo.exe/V6.0)
:
callback range for write(DB handle 184, segment #4, cluster #2,
  0, 8)
Mon Oct 07 15:38:32 2002 DEBUG 0357 [04] Callback Thread
  (tucker) client #32
(1.17.1516.3453103059.1875883738.IP@192.168.105.4,
  51025/mo.exe/mo.exe/V6.0)
:
callback range for write(DB handle 184, segment #4, cluster #2,
  0, 8) -> POSITIVE
```

Comment on trace file

Client #32 gives up all ownership.

Trace file

```
Mon Oct 07 15:38:32 2002 DEBUG 0354 [05] Callback Thread
  (tucker) client #34
(1.17.1156.3453103062.1875883738.IP@192.168.105.4,
```

```
  51025/mo.exe/mo.exe/V6.0)
:
callback range for write(DB handle 184, segment #4, cluster #2,
  0, 8)
Mon Oct 07 15:38:32 2002 DEBUG 0355 [04] Callback Thread
  (tucker) client #34
(1.17.1156.3453103062.1875883738.IP@192.168.105.4,
  51025/mo.exe/mo.exe/V6.0)
:
callback range for write(DB handle 184, segment #4, cluster #2,
  0, 8) -> NEGATIVE
```

Comment on trace file

Client #34 refuses to give up ownership because it has the page read-locked. Therefore, client #35 must wait until client #34 ends its transaction and releases the lock. At this point, if you ran the ossvrstat utility, it would display a heading labeled `Client connections waiting for a lock`, under which you would see that client #35 was waiting for a lock held by client #34.

Interpreting trace files

The following notes provide additional information for help in interpreting the trace files:

- `POSITIVE` means that ownership was successfully called back, and `NEGATIVE` means that the cache manager was unsuccessful in calling back ownership.
- `DB handle 184` is assigned when the database is first opened by the client. This handle is used to refer to the database in all subsequent messages about the database until the database is closed by the client.
- `segment #4, cluster #2` identifies by number a particular segment within the database and a particular cluster within that segment. You can use `ossize` to find what objects are in the cluster, as described in ossize on page 238.
- 0,8 is a reference to the starting sector and the number of sectors. Sectors are 512 bytes. On this client platform, a page is 4 KB, so 0,8 means that the first page of this cluster is locked.
- `nl=-1` refers to the transaction nesting level. A nesting level of `-1` means that this operation refers to the current transaction, not a parent of a current nested transaction.

Debugging an ObjectStore Server on Sun Clusters

When looking for debug information about the ObjectStore server on Sun Clusters, you can find error information in `/var/adm/messages` on *any* node of the cluster. To debug an ObjectStore server that is managed by Sun Clusters, you must use the following procedure to run the server in the foreground. This procedure will also enable you to collect trace output from the server so you can redirect it to a file.

1. Find which cluster node the osserver process is currently running on. Execute the following steps as root on that node.
2. Execute the following command to disable the resource and take down the `osserver` process. Substitute your logical host name for *`hostname`*.

```
/usr/cluster/bin/scswitch -n -j EXLNosserver-hostname-hars
```

3. Set the `OS_ROOTDIR`, `LD_LIBRARY_PATH`, and `OS_SERVER_OUTPUT_LOG_NAME` environment variables.

4 Execute the following command to restart the server in the foreground:

```
$OS_ROOTDIR/lib/osserver -F -hostname hostname \
  -p $OS_ROOTDIR/etc/hostname_server_parameters
```

For additional information about debugging the server, see Debugging the ObjectStore Server on page 67.

Debugging the Cache Manager

When the cache manager encounters a severe error, it records that information in its text output file. Examine the relevant file for error messages:

- UNIX: `/tmp/osc6_out`
- Windows: `%OS_TMPDIR%\OSCMGR6.TXT`

Note that these locations are the default locations, but they can be changed during installation. Check the correct file locations for your installation.

On UNIX, verify the `setuid root`. If the cache manager fails to start, you can try starting it by hand with additional arguments to enable debugging output. On UNIX, as `root`, enter

```
oscmgr6 -d debug_level
```

On Windows, in a console window, enter

```
oscmgr6 -con -d debug_level
```

The cache manager sends debug printouts that describe its actions to standard output. The number after `-d` is the debug level. The debug level is an integer from `0` to `50`; the greater the number, the more debug information displayed.

You can confirm that the cache manager has started successfully by using the `oscmstat` command.

Debugging the Cache Manager Autostart

When an ObjectStore client is started, the application checks to see whether the cache manager is running. If the cache manager is not running, the client starts it. This process is called the *cache manager autostart*. If this process fails, you might see a message such as the following:

```
Miscellaneous ObjectStore error
Reached end-of-file reading initial message from cache manager
(PID=0) (err_misc)
Segmentation Fault
```

The resource affected is the cache manager. To obtain more information, you can do the following:

- Check the contents of the cache manager output file to see if the cache manager is running.
  - UNIX: `/tmp/osc6_out`
  - Windows: `%OS_TMPDIR%\OSCMGR6.TXT`

- Check cache and commseg file locations and dates. You might have deleted these files but did not use the `oscmrf` utility.
- On UNIX, verify `suid root` permissions on the `oscminit6` executable.

  If you are mounting the file system that contains the `oscminit6` executable, make sure that you do not specify `-o nosuid`. When you mount the file system, `-o suid` is the default; you need not specify it. If the mount is in the `fstab` file, ensure that `nosuid` does not appear in the mount options field.
- On UNIX, verify permissions on the cache and commseg files.

Alternate Technique to Debug the Cache Manager Autostart

Another technique for debugging problems with the cache manager autostart is to set the environment variable `OS_DEBUG_CMGR_STARTUP` to `1` before starting a client application that launches the cache manager. You can determine whether your application is interfering with cache manager autostart by using `ossize` to launch the cache manager instead.

Environment Variables for Debugging

The following are environment variables useful for debugging ObjectStore clients, servers, and cache managers. Most of these variables are enabled with a value of `1`.

Environment Variable	*Description*
`OS_DEBUG_NETWORK`	Displays information about each network call.
`OS_DEBUG_DYNAMIC_NET`	Displays information about network initialization. Generates debug information regarding the server's decision about the network protocol (such as TCP/IP) to use at startup.
`OS_DEF_EXCEPT_ACTION`	Controls the outcome if an unhandled exception is signaled. Useful in debugging unhandled exceptions. You can specify the following values: • `abort` — ObjectStore aborts the process. On UNIX, ObjectStore creates a core file if you are not running in a debugger. • `integer` — Specify an integer greater than or equal to `1`. Object Store exits from the program with that specified integer as the return value. • `kill` — ObjectStore action varies by platform: On Windows: `ExitProcess (0x006600);` On UNIX: `kill (getpid(), SIGKILL);`
`OS_DEF_BREAK_ACTION` Windows only	Sets a hard-coded breakpoint that allows you to obtain a stack trace before the unwinding of the stack.
`OS_DEF_MESSAGE_ACTION` Windows only	Determines the default message action for `_ODI_message` (used for unhandled TIX exceptions). You can set this variable to `stderr` or `stdout` to send the information there, or you can specify the name of a file to receive the information.

Environment Variables for Debugging Client Applications

The following environment variables apply only to client applications:

Environment Variable	*Description*
`OS_DEBUG_CMGR_STARTUP`	Displays information about a client's attempt to autostart the cache manager. Set to any value to display useful debug information.
`OS_DEBUG_LOCATOR_FILE`	Takes values `1`, `2`, or `3` to generate varying degrees of output from the locator file mechanism. The higher the value, the more comprehensive the reported information.
`OS_TRACE_MISSING_VTBLS`	Useful when you are debugging problems with missing virtual function tables (vtbls). This environment variable helps determine those classes that need vtbls. Any nonempty value except `0` enables the debugging message. The higher the value, the more comprehensive the reported information.
`OS_DEBUG_C0000005` Windows only	Instructs ObjectStore to display a message box if a fault occurs and the ObjectStore exception handler sees it. The message explains the fault as an attempt to read location 0x1234 from EIP 0x10231023. This can be useful in tracking down access violations in your code.

No Handler for Exception Error

ObjectStore reports errors by signaling TIX exceptions. If an unexpected error occurs, it is likely that your program does not have a handler for that exception, and your program will exit with the message `No handler for exception` followed by information on the particular error condition that was signaled. Examples of these kinds of errors appear in the paragraphs that follow.

Database error

While trying to access a database, you might receive this message:

```
No handler for exception:
ObjectStore internal error
An incorrect vector header was found during inbound relocation
at 0x30000070. The source architecture is an Intel platform with
Windows NT or Win32s and Visual C++.  The address corresponds to
offset 0x70 within cluster #0, segment #0 of database
"K:\work\bad.db".
(err_internal)
```

The resource affected is the database or the client application. To obtain more information, you can do the following:

- Use the `osverifydb` and `ossize` utilities to check all databases that the application accessed; see osverifydb on page 258 and ossize on page 238. Check both schema and production databases.
- Use a debugger to find the line of code that caused the error.
- Rebuild the database, checking for corruption as it is rebuilt.

Cannot open application schema

While trying to access a database, you might receive the following message:

```
No handler for exception:
Cannot open the application schema
```

```
<err-0025-0311> The application schema database db.adb was not found
(err_cannot_open_application_schema)
While trying to open the application schema database.
```

The resource affected is the database. To obtain more information, you can

- Verify that a current application schema database exists. It might be missing.
- Use the `ossetasp` utility (see ossetasp on page 221) to verify that the executable has the correct location of the application schema database.

When you move an application to a machine that is not running a server, leave the application schema database on the server host. You must then run the `ossetasp` utility to patch the executable so it can find its application schema database.

The application schema database must be local to a server. The only exception to this is when you use a locator file. See Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 265.

Invalid address

When trying to run a client application you might receive a message such as the following:

```
No handler for exception
ObjectStore internal error
ObjectStore referenced invalid address 0x7fff0028(0)
```

The resource affected is the client application. To obtain more information, you can

- Execute the program in the debugger and determine the line of code that has a bad pointer.
- Obtain a back trace from a debugger. It might be that the program is using ObjectStore features incorrectly.

This is a case in which you might not be able to figure out the problem but you can obtain enough information for Technical Support to solve the problem quickly.

Incompatible releases of ObjectStore

If you are running a Release 6 server and a non-Release 6 server on the same machine, you can receive the following message when trying to run an application:

```
Attempt to use unsupported server protocol.
<err-0025-1175> ObjectStore server on host "tucker" cannot be
used because it uses protocol version 1.69, but this client uses
version 1.70
(err_protocol_not_supported)
```

ObjectStore displays this message when the server is an older version of ObjectStore Release 6 than the client. You can use a newer server with an older client, but you cannot use a newer client with an older server. If you are using a pre-release 6 server with a release 6 client, ObjectStore displays the following message:

```
Attempt to use unsupported server protocol.
<err-0025-1174> ObjectStore server on host "tucker" cannot be
used because it is older than ObjectStore release 6.0.
(err_protocol_not_supported)
```

Related to the above error messages is the following, which can occur when a release 6 client attempts to use a pre-release 6 database:

```
The file database appears to be in an inconsistent state
>> tucker:K:\work\test.db -> <err-0019-0003> The database
"DB handle 170, File DB: K:\work\test.db" cannot be used
by a client in ObjectStore release 6.0 or newer.
(err_inconsistent_db)
```

Changes not propagated to database

Either of the following two error messages can occur if the database or the transaction log was modified, renamed, deleted, or moved other than by Objectstore, with the result that changes that should have been propagated to the database were lost:

```
The file database appears to be in an inconsistent state
tucker:K:\temp\rtee\rteeMain.db -> <maint-0019-1211> Creation
of database File DB: K:\temp\rtee\rteeMain.db was started but
not committed.
(err_inconsistent_db)
```

```
The file database appears to be in an inconsistent state
tucker:K:\temp\rtee\rteeColl.db -> <maint-0019-1212> Committed
changes to database File DB: K:\temp\rtee\rteeColl.db were not
propagated from the transaction log to the material database.
(err_inconsistent_db)
```

Cannot commit

You must ensure that when two or more ObjectStore servers receive modified data in a single transaction, each server can communicate with each of the other servers. If all servers involved in such a transaction cannot communicate with each other, ObjectStore aborts the transaction with `err_cannot_commit`.

The exception occurs only when a two-phase commit has problems. A two-phase commit occurs only if multiple servers are being written to in that transaction. If a client reads from ServerA in one transaction and writes to ServerB in the same transaction, it is not a two-phase commit and it is not necessary for ServerA and ServerB to be able to communicate across a common network.

Before Calling Technical Support

When you contact Technical Support, be prepared to provide the following information:

- Site ID number
- Hardware platform
- Operating system and release number
- ObjectStore release number
- Severity of the problem
- Description of the problem
- Compiler information (if applicable)
- Information found in the server or cache manager output file (if this is relevant)
- Server debug mode output if there is a problem with the server
- Debugger output if there is a problem with an application (stack trace)
- Test case

Chapter 2
Server Parameters

This chapter describes server parameters, which determine some aspects of the way the ObjectStore server functions. There are defaults for all parameters. You need not do anything to use the defaults.

To modify a parameter, see the instructions in Chapter 7, Managing ObjectStore on UNIX, on page 299; or Chapter 8, Managing ObjectStore on Windows. After you modify a server parameter, you must shut down and restart the server for the change to take effect.

Server parameters are available on all platforms unless otherwise noted. Case is not significant.

List of Server Parameters

Server parameters described in this chapter include those in the following list.

Admin Host List

Default: none

The `Admin Host List` parameter specifies the path of a file. This file must contain a set of primary names of hosts, one per line. (If DNS is in use, they must be fully qualified domain names.) The hosts listed in this file are the hosts on which an administrative user, designated with the `Admin User` parameter, can perform ObjectStore server operations.

When this parameter is not set, an administrative user can perform ObjectStore server operations on any ObjectStore host.

If you specify a host in the file pointed to by the `Admin Host List` parameter, that host requires only `SYS` authentication. This is true regardless of the setting of the `Authentication Required` server parameter for that host. When you create an administrative hosts list, and you also have a host access list set with the `Host`

Access List server parameter, ObjectStore adds all hosts in the administrative hosts list to the host access list.

Admin User

Default: none

When Admin User is set to a user name, it allows that user to run any ObjectStore utility without having root permission. When this parameter is not set, the user must have root permission to run ObjectStore utilities, for example, ossvrshtd and ossvrclntkill.

When you set this parameter, you designate a user who does not have root privileges to perform ObjectStore administrative functions. This administrative user can execute any ObjectStore utility that a user with root permission can, except operating on file databases. File databases are not affected when the Admin User parameter is set.

When a utility operates on rawfs databases, administrative users are not restricted by access control. They can access any database and perform any operation even though they are operating under their own user names and groups.

When a utility operates on file databases, administrative users have the same restrictions they normally have. Being an administrative user does not give them additional privileges.

Allow NFS Locks (UNIX Only)

Default: yes

When the Allow NFS Locks parameter is set to yes (the default), the server can perform database-level locking. If database-level locking is on, when an ObjectStore server handles access to a remote (or local) database, it holds a lock on the database as long as the database is open. If the database is open for read-only, the server holds a read lock; if the database is open for read or write, the server holds a write lock.

When an ObjectStore server holds a read lock on a database, other servers are prevented from acquiring a write lock on the database. Read access by other ObjectStore servers is allowed. A read lock does not prevent write access by another application if the same server handles the access.

When an ObjectStore server holds a write lock on a database, other servers are prevented from acquiring either a read lock or a write lock on the database. Again, this does not prevent access by another application if the same ObjectStore server handles the access.

If a server is blocked from acquiring a lock, ObjectStore signals the exception err_database_lock_conflict. Access is not retried automatically.

Caution

You can turn off database-level locking by setting the server parameter Allow NFS Locks to no. You must use extreme caution if you turn off locking. Concurrent database access by different servers can corrupt the database. Note that a mistake in the contents of a locator file could cause unintentional concurrent access of this sort.

Windows — File locking is always turned on.

Allow Remote Database Access

Default: `no`

The `Allow Remote Database Access` parameter determines whether the server can handle access to remote databases. If ObjectStore determines from a locator file that a particular ObjectStore server should handle access to a database remote to that server, and the value of this parameter is `yes` for that server, the server handles access to the database. If the value is not `yes` for the server, the exception `err_file_not_local` is signaled.

If you allow any server-remote access, each database should be assigned exactly one ObjectStore server that handles all access to it by all applications, unless that database is never opened for read or write. When one server handles access to a database, it can prevent concurrent access by other ObjectStore servers. See also `Allow NFS Locks (UNIX Only)` on page 79 and Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 265.

Caution — Use this parameter with caution. A mistake in the content of a locator file could result in unintentional concurrent access.

Allow Shared Communications (UNIX Only)

Default: `yes`

The `Allow Shared Communications` parameter controls whether the server allows shared memory communications between itself and the client when they are on the same host. Allowing such communications improves performance. It is a Boolean value that defaults to `yes` (meaning shared memory communication is enabled). If these communications are enabled, the server and client exchange data by mapping the client's cache into the server's virtual memory.

It is possible for the server to run out of virtual memory if you have `Allow Shared Communications` set to `yes` and you have many local clients. This can happen because the server maps each local client's cache information into its virtual memory. Setting `Allow Shared Communications` to `no` eliminates the problem.

Authentication Required

The defaults are

- `NONE` on Windows 98
- `Name Password` on Windows NT
- `SYS` on other platforms

The `Authentication Required` server parameter specifies the way the server controls database access.

How the server controls access to data — ObjectStore has a client/server architecture. When an application reads or modifies a database, it sends messages to an ObjectStore server, which in turn reads and writes the data in the database.

Because each ObjectStore server handles requests from many different users, it is responsible for enforcing access control to files containing databases. It must use privileged access to read or write any user's database, and it must ensure that only users entitled by the host system's rules for access control are allowed access to databases. To implement access control, the ObjectStore server must know the user name and group of the client process that is requesting that it operate on a database.

Authentication

If you start an application that asks the ObjectStore server to read or modify a protected file, the server determines whether you have proper read/write permission for the file. This access checking has two parts. The first part is *authentication* — the operation in which the server learns who the client is and on behalf of which user it is performing operations. The second part is verifying whether that user has permission to perform the requested operation.

The type of authentication ObjectStore uses is determined by the value of the server parameter `Authentication Required`.

Note that certain server operations that deal with file databases require authentication. If the client does not provide authentication, the server refuses to perform such an operation. These operations include looking up, creating, and deleting file databases, or asking their size or access modes.

Administrative commands for which authorization is required send authentication to the server first. Administrative commands are ObjectStore utilities that affect other people's work environment, for example, the utility that shuts down the server.

ObjectStore applications send authentication to the server the first time the application performs an operation on that server for which authentication is required. After an application sends authentication information to a server, it need not do so again for the lifetime of the process.

Administrative operations for authorized clients

The server performs several administrative operations only on behalf of an authorized client. These are the server operations invoked by the `ossvrshtd` and `ossvrclntkill` utilities, with these exceptions:

- `ossvrshtd` is also allowed for the user who owns the server process.
- `ossvrclntkill` is allowed if the client thread to be killed is owned by the user issuing the command.

If the `Authentication Required` server parameter is set to `NONE` (see the following), all clients are authorized to perform administrative operations.

Parameter values

The `Authentication Required` server parameter specifies the preferred way the server controls database access. There are six possible types of authentication:

- `NONE` — The only available option on Windows 98, which does not support security.

  On UNIX platforms, clients who present authentication `NONE` cannot access file databases. Clients always present some other form of authentication, if possible.

On Windows NT, clients who present authentication NONE can access file databases. Clients can access any file accessible to the user who is running the server (usually Local System).

- SYS — UNIX only. ObjectStore uses the Sun ONC RPC AUTH_SYS authentication method.

  The client sends the server a UNIX user ID and group ID, which the server trusts. The ObjectStore client library always sends the current effective user ID and group ID. This is the same mechanism used by the NFS protocol. See Accessing UNIX Databases from Windows on page 323.

  To allow a Windows NT client to provide SYS authentication to a UNIX server, you must set the UNIX.UID (user ID) and UNIX.GID (group ID) entries in the Windows NT registry.

  To prohibit Windows NT clients from returning SYS authentication to UNIX servers, you can either set Authentication Required to Name Password or forbid the user ID and group ID entries in the registry.

Caution

If you use this method, be aware of the following points:

- If untrustworthy users have access to the root password, they can assume the identity of any user.
- A malicious user might contrive to patch the ObjectStore client so that it sends some access identity information other than the current effective user ID and group set.
- Not all systems can generate this information in a client. If you run an ObjectStore server on such a system, this method is not available.

Windows 98 clients do not support SYS.

- DES — Sun and System V.4.0 only. ObjectStore uses the Sun ONC RPC AUTH_DES authentication, also known as *secure RPC*. To enable this mechanism, you must first set up the Network Information System (NIS), run the keyserv daemon on all client and server hosts, and register public keys in the publickey NIS map. (See the newkey, chkey, keyserv, and keylogin manual pages.)

  For a full explanation of the secure RPC mechanism, see the documentation supplied with your operating system.

  DES authentication protects against abuse of the root password and against straightforward attempts to patch an ObjectStore client to send a fraudulent access ID. However, it is not secure against a determined intruder, due to bugs both in its design and in the reference implementation.

  DES is supported by SYS V-based UNIX platforms, and it must be installed and configured on the client and server. ObjectStore cannot determine whether the DES installation is correct; it simply attempts to use it.

- NT Local — Windows NT only. Allows Windows NT clients to be authenticated by local servers using ImpersonateLogged-OnUser authentication.

- `NT Remote` — Windows NT only. Allows Windows NT clients to be authenticated by local and remote servers using `Security Service Provider` authentication.
- `Name Password` — All platforms except Windows 98. ObjectStore requires each client application to send a user name and password to the server, which validates them. See "User interface to authentication" on page 84. The advantage of this method is that there is no way to fool the server — it grants exactly the access available to the user it authenticates. The disadvantage is that ObjectStore cannot arrange a *trusted path*. That is, there is no way that a user, prompted for a password, can be sure that the password will be sent to the ObjectStore server and only to the ObjectStore server. A malicious program author could solicit and retain passwords.

  On Windows, `Name Password` works only when the server is run as a service. When the server is running as a normal process, `Name Password` is omitted from the list of allowed types. This is a Windows restriction.

Parameter value might imply a set of values

When you set the parameter to one of the authentication types listed next, the server can use that value as well as any values that follow it in the list, if the value can be specified on that server's platform.

```
NONE
SYS
DES
NT Local
NT Remote
Name Password
```

To determine the list of authentication types that a server supports, run the `ossvrstat` utility. See ossvrstat on page 248 for further information.

How does a client choose?

How does a client determine the type of authentication to provide to the server? The client tries to match the authentication type it supports with the type of authentication supported by the server as described in the following steps:

The client examines the `OS_AUTH` environment variable. If it specifies an authentication type the server supports, the client uses that type. See `OS_AUTH` on page 105.

1. The client uses one of the programmatic authentication types (`SYS`, `DES`, `NT Local`, or `NT Remote`) that both it and the server support.
2. The client uses `NONE` if the server allows it.
3. The client uses `Name Password` if the server allows it.
4. If no authentication method can be found, `err_authentication_failure` is signaled.

The following matrix shows the order in which various combinations of clients and servers attempt to match authentication types. An x indicates that an authentication type will not work for that combination.

UNIX Client / UNIX Server	*UNIX Client / NT Server*	*NT Client / UNIX Server*	*NT Client / NT Server*	*Authentication Type*
1	x	1	x	SYS
2	x	x	x	DES (on platforms that support DES)
x	x	x	1	NT Local
x	x	x	2	NT Remote
3	1	2	3	NONE
4	2	3	4	Name Password

Windows NT clients

When a Windows NT client provides authentication, it must provide the domain name, the user name, and the user password.

Suppose that a Windows NT client contacts a UNIX server that can accept SYS authentication. Further suppose that user ID and group are set in the Windows NT registry. In this case, the client provides SYS authentication. If the server does not allow SYS authentication or if the user ID and group are not set in the registry, the client provides the first type of authentication that it supports in the list of server-supported authentication types. In some cases, this is Name Password. It might, however, be NONE, which means that a Windows NT client can access file databases that are accessible by the user running the server.

Exceptions

When a server is using Name Password for authentication and detects something wrong, it signals the err_authentication_failure exception. This usually means that there is no user by the specified name or that the password is incorrect.

If the server receives a command that needs authentication but the connection has not been authenticated, and Authentication Required is not NONE, the server rejects the command and signals the err_rpc_auth_tooweak (client credential too weak) exception. This means that the client could not provide the authentication that the server requested. This happens when a client cannot generate any of the types of authentication that the server requires.

User interface to authentication

By default, the interface to Name Password authentication communicates with the user interactively. On UNIX, the application interface uses the /dev/tty device or the stdin and stdout streams to prompt for a user name and a password. On Windows, console applications work the same way. Windows applications pop up a dialog box. Your application can control the Name Password interface with the functions objectstore::set_simple_auth_ui() and get_simple_auth_ui(), which are described in the *C++ API Reference*.

Cache Manager Ping Time

Default: 300 seconds

The `Cache Manager Ping Time` parameter specifies a number of seconds. Whenever the time between `Cache Manager Ping Time` seconds and twice `Cache Manager Ping Time` seconds has elapsed with no message from the client, the server attempts to send a message to the client's cache manager to determine whether the client is still active. If the cache manager cannot be contacted or the cache manager indicates that the client does not exist, the server disconnects the client connection. The minimum number of seconds you can specify is 10.

A large value for this parameter reduces network traffic and reduces the chance that a client will be disconnected because of a transient network failure. A small value increases network traffic but ensures that a client disconnected from the network cannot hold locks for more than a short period of time.

If `Cache Manager Ping Time` and `Cache Manager Ping Time In Transaction` have different values, ObjectStore uses `Cache Manager Ping Time` when the client is not in a transaction.

Cache Manager Ping Time In Transaction

Default: 300 seconds

The `Cache Manager Ping Time In Transaction` parameter is the same as the `Cache Manager Ping Time` parameter, except that the server uses the interval you specify only during transactions. The number of seconds you specify for `Cache Manager Ping Time In Transaction` must be less than or equal to the value set for `Cache Manager Ping Time`. The minimum you can specify is 10 seconds.

If `Cache Manager Ping Time` and `Cache Manager Ping Time In Transaction` have different values, ObjectStore uses `Cache Manager Ping Time In Transaction` when the client is in a transaction.

Cluster Growth Policy

Default: `512 20K, 1K 40K, 2K 80K, 4K 160K, 8K 320 K, 16K 640K, 32K`

The `Cluster Growth Policy` parameter specifies the policy for growing clusters. This policy is applied when a cluster grows because of persistent object allocation. The policy is not applied when the size of a cluster is set by a utility such as `oscopy`, `osrestore`, or `oscompact`, when a huge object is created in its own cluster, or when the cluster is presized by `os_cluster::set_size()`. In those cases, the size will be exactly what was requested.

The syntax is:

```
Cluster Growth Policy: policy
policy = { growth size , }* growth
growth = integer % | size
size = integer { | K | M | G }
```

policy is a series of one or more *growth*/*size* pairs, where the last *size* is implicitly infinite. *size* is specified in bytes (the default), kilobytes (K), megabytes (M), or gigabytes (G). *integer* is a decimal integer; hexadecimal is not allowed.

When *growth* is followed by the percent sign (%), it specifies a percentage of the current size; otherwise, it specifies an absolute size. The new size of the cluster data will be rounded up to the next higher multiple of the absolute size, or the increase will be at least as large as a percentage.

ObjectStore rounds up the new size, as well as *growth* and *size* values, to the next higher multiple of units of its choosing.

Policy pairs must be listed in increasing order of size. The applicable policy is the one with the smallest size that is greater than or equal to the current size. When interpreting the policy series, read each `{ growth size }` pair as though it said "*growth* up to *size*". If *growth* is at the end of a sequence of `{ growth size }` pairs, it should be read as though it said "otherwise *growth*". For example, the first pair in the default policy should be interpreted as though it said, "Grow the cluster in increments of 512 bytes up to the size of 20 KB." By default, the maximum granularity for growing a cluster is 32 KB in the default policy.

To take the default policy as an example, if the current cluster size is 170 KB, and it needs to grow by 3 KB to accomodate newly-allocated objects, the new size will be rounded up to the next multiple of 8 KB, 176 KB, leaving 3 KB of free space at the end of the cluster that can be used for further object allocation.

Changing the cluster growth policy can be an effective way to prevent database fragmentation. For more information, see Managing Database Fragmentation on page 44.

Note

In releases before ObjectStore 6.1, the default policy was 512.

Database File Growth Policy

Default: `4K 40K, 10% 10M, 1M`

The `Database File Growth Policy` parameter specifies the policy for growing database files. This policy is applied when a file is created or grows. It is not applied when the file is presized by `os_database::set_size()` or `os_database::set_size_in_sectors()`.

The syntax is:

```
Database File Growth Policy: policy
policy = {growth size , }* growth
growth = integer % | size
size = integer { | K | M | G | T }
```

policy is a series of one or more *growth*/*size* pairs, where the last *size* is implicitly infinite. *size* is specified in bytes (the default), kilobytes (K), megabytes (M), gigabytes (G), or terabytes (T). *integer* is a decimal integer; hexadecimal is not allowed.

When *growth* is followed by the percent sign (%), it specifies a percentage of the current size; otherwise, it specifies an absolute size. The new size of the database file will be rounded up to the next higher multiple of an absolute size, or the increase will be at least as large as a percentage.

ObjectStore rounds up the new size, as well as *growth* and *size* values, to the next higher multiple of units of its choosing.

Policy pairs must be listed in increasing order of size. The applicable policy is the one with the smallest size that is greater than or equal to the current size. When interpreting the policy series, read each `{ growth size }` pair as though it said "*growth* up to *size*". If *growth* is at the end of a sequence of `{ growth size }` pairs, it should be read as though it said "otherwise *growth*". For example, the first pair in the default policy should be interpreted as though it said, "Grow the file in increments of 4KB up to the size of 40KB." By default, the maximum granularity for growing a file is 1 M in the default policy.

Changing the database file growth policy can be an effective way to prevent database fragmentation. For more information, see Managing Database Fragmentation on page 44.

DB Expiration Time

Default: 300 seconds

The `DB Expiration Time` parameter specifies the number of seconds that the server keeps a rawfs database open after the last user has closed the database. This allows data propagation to happen in the background, which means that users do not have to wait for propagation to occur before they can close the rawfs database. Also, applications that use the rawfs database start more quickly if they are starting during `DB Expiration Time`.

Deadlock Victim

Default: `Work`

The `Deadlock Victim` parameter specifies the method that the server uses to select a victim in the event of a deadlock. Specify one of the following to determine the client that is the victim. The server sends a message to the selected client to abort the transaction.

- `Age`
  The youngest client. A client's age is calculated as the time since its last successfully committed transaction.
- `Current`
  The client that made the last request to the server, causing the server to detect the deadlock.
- `Oldest`
  The oldest client. A client's age is calculated as the time since its last successfully committed transaction.

- `Random`
  A random client.
- `Work`
  The client that has done the least amount of work, as measured by RPC calls to the server during the current transaction. This is the default.

This method of choosing a victim is secondary to transaction priority. See `objectstore::set_transaction_priority()` in the *C++ API Reference* for more information on transaction priorities.

Deadlock occurs when two or more processes are waiting for locks and there is a circular dependency such that none of them can obtain the locks. For example, ProcessA is waiting for a lock held by ProcessB, which is waiting for a lock held by ProcessA. Because the dependency is circular, none of the processes can obtain the requested lock. If the ObjectStore server did not detect deadlock situations, the processes involved would wait for an infinite length of time.

When deadlock occurs, the server detects it automatically and chooses a victim whose transaction is aborted. If the aborted transaction is a lexical transaction, it is retried automatically. If it is a dynamic transaction, the transaction is aborted with `err_deadlock`.

When the server aborts lexical transactions, it rolls back the persistent data that has been modified during the transaction to its pretransaction state. Also, stack variables declared within the scope of the transaction go out of scope. Special care must be taken for stack variables whose values have been changed within the transaction and for space allocated on the heap during the transaction.

To obtain information about where the deadlock occurs, run the server in debug mode.

Direct to Segment Threshold

Default: 128 sectors (64 KB)

If the `Direct to Segment Threshold` parameter specifies a value of 4194304 sectors (2 GB) or greater, the server is disabled from writing segments directly to the database; that is, the data will always be written to the log before being written to the database.

The following information applies to clients previous to Release 6.0.

The `Direct to Segment Threshold` parameter specifies, in sectors, the threshold value that determines whether the server writes segments first to the log and then to the database, or writes them directly to the database. If less than this threshold is written past the current end of a segment during a transaction, the data is written to the log before being written to the database. If the number of sectors written is greater than this threshold, the data is written directly to the database.

Choosing a value depends on the size of the log and the cost of writing and flushing the data separately from writing and flushing the log record. For example, if you need to add 1 KB to each of 100 segments, writing 1 KB to each segment accesses the

disk 100 times. If average disk speed is between 8 and 12 milliseconds per access, it would take between 800 milliseconds and 1.2 seconds to perform all the write operations. If, however, you write the data to the log, either in a record or in the log data segment, the write operation would probably be to contiguous disk storage and take only about 50 milliseconds. This parameter lets you choose between

- Faster access and a larger log file
- Slower access and a smaller log file

The server applies the following tests in the following order; the first one that matches determines the way the data is stored:

1. If the data is being written to a newly created segment, data is always written directly to the database segment.
2. If the following conditions are met, the data goes directly to the database (this formula means that the decision to write data directly to a database is not changed by the size of individual write operations).
   - Data is being written past the current end of the database segment.
   - The number of the last sector being written minus the current committed size of the segment is greater than the new `Direct to Segment Threshold`.
3. If neither of the previous conditions applies, data is put in the log until the commit is done.

ESTALE Implies Database Deleted (UNIX Only)

Default: `No`

When the `ESTALE Implies Database Deleted` server parameter is set to `yes`, ObjectStore allows the occurrence of events that could corrupt the database if NFS gives an `ESTALE` status, instead of reporting errors that could crash the server. This server parameter is effective only if the `Allow Remote Database Access` parameter is set to `yes`; see Allow Remote Database Access on page 80. The default for `ESTALE Implies Database Deleted` is `no`.

Failover Heartbeat File

Default: none

The `Failover Heartbeat File` server parameter specifies the pathname of the heartbeat file, making it accessible to a failover server pair — that is, to the primary and secondary servers in the pair. This parameter must be specified whenever the following are true:

- ObjectStore-managed failover is enabled; for more information, see ObjectStore-Managed Failover on page 286. The heartbeat file is not needed when failover is managed by a cluster operating system.
- You are not using rawfs or the heartbeat is to be stored in an operating system file rather than in rawfs.

See Chapter 6, High Availability of Data, on page 285 for more information about setting up a failover system.

Failover Heartbeat Time

Default: none

The `Failover Heartbeat Time` parameter must be specified if you are using a failover server. This parameter can be set to between 2 and 60 seconds. The parameter defines how often a heartbeat message is written to disk. In the event of a failure, it takes five times the `Failover Heartbeat Time` for the secondary server to recognize the failure and to take over.

This server parameter can only be used with ObjectStore-managed failover. For more information, see ObjectStore-Managed Failover on page 286.

Failover Script

Default: none

The `Failover Script` parameter specifies the path of a script or program that should be executed when the secondary server takes over from the primary server in an ObjectStore failover system. The script or program should return `0` if successfully executed. If a nonzero value is returned, the secondary server will terminate.

This server parameter can only be used with ObjectStore-managed failover. For more information, see ObjectStore-Managed Failover on page 286.

Host Access List

Default: none

The `Host Access List` parameter specifies the path of a file. This file must contain a set of primary names of hosts, one name per line. (If DNS is in use, they must be fully qualified domain names.) The server refuses connections from any host not on the list, or whose name cannot be determined. This mechanism is only as secure as the available means for translating host addresses to host names. On some networks, there might not be a secure method.

This parameter is intended for use in environments in which a machine is on a network with untrustworthy hosts and `DES` authentication is either unavailable or unworkable.

When you create a host access list and you also have an administrative hosts list set with the `Admin Host List` server parameter, ObjectStore adds all hosts in the administrative hosts list to the host access list.

Identical Pathnames on Failover Server

Default: none

The `Identical Pathnames on Failover Server` parameter identifies which files can be used without risk of losing access to them when a failover occurs. A server running with failover turned on will allow these files to be used for databases, transaction logs, and rawfs partitions, but will not allow any other files to be used. Attempting to use another file causes an exception with the message:

file is not declared to be shared between the two servers in this failover pair

`Identical Pathnames on Failover Server` specifies a comma-separated list of strings. Whitespace after a comma is ignored, and string comparison is case-insensitive. If a file pathname begins with a specified string, the file is allowed to be used. Any file with a matching pathname must be available to both servers in the failover pair and must have the same pathname on both machines.

Following is a Windows example of the `Identical Pathnames on Failover Server` parameter. It specifies a shared disk for databases is mounted as `X:` on both machines and a shared disk for the transaction log is mounted as `Y:` on both machines:

```
Identical Pathnames on Failover Server:   X:, Y:
```

Following is a UNIX example of the `Identical Pathnames on Failover Server` parameter. It specifies a shared disk is mounted on `/shared/ostore` on both machines:

```
Identical Pathnames on Failover Server:   /shared/ostore/
```

Any file whose pathname starts with a string that is a member of the `Identical Pathnames on Failover Server` list will be assumed by the server to be accessible to both servers in the failover pair and to have the same pathname in both servers. If the string ends with a directory separator, such as `/` on UNIX, the entire subtree of directories under the named directory is assumed to be shared by the two servers in the failover pair. If the string does not end with a directory separator, it is as if the wildcard character (`*`) were appended to the string; thus `/shared/ostore` would match the subtrees under both `/shared/ostore-databases` and `/shared/ostore-logs` if those were the names of two directories.

If the `Identical Pathnames on Failover Server` parameter is not specified and the server is a failover server, then only rawfs databases can be used, the transaction log must be in rawfs, and all rawfs partitions must be raw-disk (that is, not in an operating-system file).

Note

`Identical Pathnames on Failover Server` always applies to entire directory subtrees under the specified locations. ObjectStore does not validate that all files within the subtree can be shared and are accessible by failover servers. It is the user's responsibility to ensure that no mount point in the subtree mounts an unshared file system.

See Chapter 6, High Availability of Data, on page 285 for more information about setting up a failover system.

Log Data Segment Growth Increment

Default: 2048 sectors (1 MB)

The `Log Data Segment Growth Increment` parameter specifies the number of sectors to add to the data segment in the transaction log when more room is needed.

Log Data Segment Initial Size

Default: 2048 sectors (1 MB)

The `Log Data Segment Initial Size` parameter sets the initial size of the data segment in the transaction log in sectors.

Log File

Default: rawfs

The `Log File` parameter specifies the path of the transaction log file. If you do not specify a value for `Log File`, the server maintains the log in the rawfs, where it does not need a name. If you do not have a rawfs and you do not specify a value for `Log File`, the server fails to start and displays the following message:

```
There are no partitions specified in the parameters file. Whenever
partitions are omitted from the parameters file a log file path must
be specified in the parameters file.
```

If you have a rawfs, you do not specify this parameter. See The Server Transaction Log on page 27.

If you specify this parameter, the path name cannot be for a raw partition. The directory that contains the log file must exist before you start the server. When the log is in the native file system, you should place it where it can never be deleted by a user accidentally. You might want to name it in such a way that it is never deleted.

For best performance, the log file should be on a disk that is not used for any other purpose.

Caution

Deleting the log file can cause database corruption.

Windows

On Windows, the value of this parameter is normally set while you run the ObjectStore Setup utility. If you indicate that you do not want the log file to be in the rawfs, the utility prompts you to enter the name of the log file. You can press Enter to select the default. On Windows, the default log file name is `OSSVXNT.LOG`.

Log Record Segment Buffer Size

Default: 1024 sectors (512 KB)

The `Log Record Segment Buffer Size` parameter defines the amount of buffer space reserved for log records.

Log Record Segment Growth Increment

Default: 512 sectors (256 KB)

The `Log Record Segment Growth Increment` parameter specifies the number of sectors by which to increase the log record segment when more room is needed.

Log Record Segment Initial Size

Default: 1024 sectors (512 KB).

The `Log Record Segment Initial Size` parameter sets the initial combined size of the two log record segments in sectors.

Max AIO Threads

Default: 3 threads

The `Max AIO Threads` parameter determines the number of threads the server can start, to perform file input/output (I/O). One thread performs all I/O for a given file. Each thread can handle I/O for multiple files. ObjectStore assigns files to threads on a rotating basis.

If you never open more than a given number of files, you might want to set this parameter to that number to ensure that there is one thread per file.

While ObjectStore does not have a limit on the number of threads you can use, your operating system might.

Max Connect Memory Usage

Default: `0` (unlimited virtual memory)

The `Max Connect Memory Usage` parameter defines in kilobytes an amount of virtual memory. When the server is using that amount of virtual memory, it refuses new connections from clients. As long as the amount of virtual memory in use is less than the amount set for `Max Connect Memory Usage`, the server allows a new connection. The server does not consider the amount of memory the new connection might need.

ObjectStore does allow its utilities to connect to the server when the amount specified for `Max Connect Memory Usage` is exceeded. However, the amount of virtual memory being used must be less than (`Max Connect Memory Usage` + `Max Memory Usage`)/2.

The `Max Connect Memory Usage` parameter is always at least 1 MB less than the `Max Memory Usage` parameter. When you increase the value set for `Max Connect Memory Usage`, the server increases `Max Memory Usage` if necessary.

Max Data Propagation Per Propagate

Default: 10% of size specifiedby the `Propagation Buffer Size` parameter but not less than 256KB and not more than 8MB.

The `Max Data Propagation Per Propagate` parameter specifies the maximum number of sectors that can be moved in one propagate operation. This limits the impact of propagation on the handling of client requests.

When counting the amount of data being propagated, ObjectStore weights the effect of noncontiguous data by 64 sectors. This means that with `Max Data Propagation`

`Per Propagate` set to 512 sectors, a maximum of eight noncontiguous writes can occur in a single propagation.

You can set the value of this parameter to more than the default in order to decrease fsync overhead. You can set the value of this parameter to less than the default in order to decrease the delays experienced by database reads that need to wait while database writes complete.

Max Data Propagation Threshold

Default: Equal to the size specified by the `Propagation Buffer Size` parameter

The `Max Data Propagation Threshold` parameter sets the number of sectors that can be waiting to be propagated. When the number of sectors waiting to be propagated exceeds one half the value specified for `Max Data Propagation Threshold`, the server forces propagation to run faster. When the number of sectors waiting exceeds this parameter, propagation takes priority over clients.

You can set the value of this parameter to less than the default in order to use some of the Propagation Buffer for caching; it should not be done to delay propagation. Setting the value of this parameter to more than the default is not useful.

Max Memory Usage

Default: `0` (unlimited virtual memory)

The `Max Memory Usage` parameter specifies in kilobytes the maximum amount of virtual memory the server can use. When the server starts, it checks this parameter, immediately allocates the specified amount of memory, and then frees it. On some platforms, this initial allocation confirms the availability of the memory and reserves swap space for the server.

Specify a number that includes the minimum needed, plus some other arbitrary amount. If you specify a number that is less than the minimum needed, the server increases the number to an appropriate value. When calculating the minimum amount of virtual memory needed, the server allows for

- Five clients (approximately 64 KB per client)
- Amount configured for message, propagation, and log buffers

The `Max Memory Usage` parameter works in coordination with the `Max Connect Memory Usage` parameter. For more information, see `Max Connect Memory Usage` on page 93.

Max Two Phase Delay

Default: 30 seconds

The `Max Two Phase Delay` parameter specifies the maximum number of seconds that the server can delay a two-phase commit so that the server can back up data. Sometimes, when ObjectStore is backing up data on multiple servers at the same time, ObjectStore must synchronize the relevant servers. This ensures that a transaction applying to more than one server is committed on all relevant servers. To

do this, the server must sometimes delay the commit of some transactions. Usually, ObjectStore can synchronize the servers in much less than a second.

If a client aborts during a two-phase commit, this value could be exceeded. If this value is exceeded, ObjectStore cancels the delay and tries again.

Message Buffer Size

Default: 512 sectors (256 KB)

The `Message Buffer Size` parameter sets the size of the message buffer that the server uses for processing messages from clients. The server reads data from this buffer and writes data requested by the clients into this buffer. The value of this parameter is rounded to a multiple of 64 KB.

N Message Buffers

Default: 4 message buffers

The `N Message Buffers` parameter sets the number of message buffers the server uses to communicate with clients. This number defines the maximum number of clients that can simultaneously perform operations that fetch or store persistent data.

Notification Retry Time

Default: 60 seconds.

The `Notification Retry Time` parameter specifies the time, in seconds, between retries for server-to-server communication. It is used during two-phase commit recovery.

Partition*N*

Default: none

The `Partition`*`N`* parameter (where *`N`* is an integer) specifies a partition in the rawfs. See Creating a Rawfs in the chapter for your platform for information about using this parameter.

Preferred Network Receive Buffer Size

Preferred Network Send Buffer Size

Default: 16384 bytes

The `Preferred Network Receive Buffer Size` and `Preferred Network Send Buffer Size` parameters specify the network buffer sizes in bytes for sending and receiving messages to and from the server, respectively.

UNIX — On many UNIX platforms, these are TCP buffer sizes.

Propagation Buffer Size

Default: Varies according to machine hardware — 8% of main memory divided by the number of processors minus 64MB per processor, but not less than 4MB and not more than the following maximums:

AIX5 and HP32	64 MB
Other 32-bit platforms	256 MB
All 64-bit platforms	512 MB

The `Propagation Buffer Size` parameter selects the amount of buffer space reserved for propagation. This space is used for holding data that are to be written to target databases. If the buffer size is large enough, data written to the log can be kept in memory until propagation to the database, thus eliminating the need to read the log.

Propagation Sleep Time

Default: 60 seconds

The `Propagation Sleep Time` parameter determines the number of seconds between propagations. This parameter takes effect only when there is data to be propagated. If a lot of data is waiting to be propagated, the server decreases this interval temporarily.

RAWFS Partition Growth Policy

Default: `1M`

The `RAWFS Partition Growth Policy` parameter specifies the policy for growing expandable rawfs partitions. This policy is applied when a partition is created or grows.

The syntax is:

```
RAWFS Partition Growth Policy: policy
policy = {growth size , }* growth
growth = integer % | size
size = integer { | K | M | G | T }
```

`policy` is a series of one or more *`growth`*/*`size`* pairs, where the last *`size`* is implicitly infinite. *`size`* is specified in bytes, kilobytes (`K`), megabytes (`M`), gigabytes (`G`), or terabytes (`T`). *`integer`* is a decimal integer; hexadecimal is not allowed.

When *`growth`* is followed by the percent sign (`%`), it specifies a percentage of the current size; otherwise, it specifies an absolute size. The new size of the rawfs partition will be rounded up to the next higher multiple of an absolute size, or the increase will be at least as large as a percentage.

ObjectStore rounds up the new size, as well as *`growth`* and *`size`* values, to the next higher multiple of units of its choosing.

Policy pairs must be listed in increasing order of size. The applicable policy is the one with the smallest size that is greater than or equal to the current size. When interpreting the policy series, read each `{ growth size }` pair as though it said "*growth* up to *size*". If *growth* is at the end of a sequence of `{ growth size }` pairs, it should be read as though it said "otherwise *growth*".

Changing the rawfs partition growth policy can be an effective way to prevent database fragmentation. For more information, see Managing Database Fragmentation on page 44.

Remote Database Grow Reserve

Default: 16 MB

The `Remote Database Grow Reserve` parameter specifies the amount of free disk space that must be available on a remote file system to grow a database. Remote databases are databases not located on the same host as the ObjectStore server. On UNIX platforms, a remote database is mounted via NFS; on Windows platforms, it is a database on a remote drive.

The `Remote Database Grow Reserve` parameter can be set to sizes between 16 MB and 1 GB. When ObjectStore tries to grow a database and the amount of free space on the remote file system is less than this parameter plus 64 KB, an `err_server_full` exception is signaled.

See Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 265, for more information about remote databases.

Restricted File DB Access (UNIX Only)

Default: `None`

The `Restricted File DB Access` parameter determines file access when an account that does not have `root` privileges starts the server. Possible values are `User`, `Group`, `All`, or `None`. When you set this parameter to a value other than `None` and the account that starts the server does not have `root` permission, ObjectStore allows access to file databases, but only by clients to which the account that started the server has `User` or `Group` access, or to all accounts if `All` is specified.

By default, `Restricted File DB Access` is set to `None`, which, in effect, disables this parameter. This means that if an account with non-`root` permission starts the server, ObjectStore allows access to rawfs databases but not to file databases.

RPC Timeout

Default: 60

The `RPC Timeout` parameter sets the timeout value in seconds for remote procedure calls (RPCs) from the server to the cache manager.

Chapter 3
Environment Variables

This chapter describes the ObjectStore environment variables, which are listed here and described in the chapter in alphabetical order.

Specifying Values for Environment Variables

You can set environment variables to modify the characteristics of the client, server, or cache manager environment. The variables you set apply to the process in which you set them and, consequently, to ObjectStore applications that you start from that process. Unless specified otherwise, these environment variables are effective only when set in the client process.

An environment variable is available on all platforms unless noted otherwise in this chapter.

The value for a variable is either an integer, a Boolean value, or a string that can have certain values as described in this chapter. When you set a variable string to an empty string, ObjectStore uses the default. When you set an integer or Boolean variable to a blank string (empty), ObjectStore interprets it as `0` or `false`.

Specifying integers

When an environment variable has a numeric value, it is an unsigned integer (leading + or – is dropped) that ObjectStore reads as a constant according to the rules of the C programming language. An empty setting results in a value of `0`. If the whole value cannot be parsed, ObjectStore signals an error such as the following:

```
<err-0001-0040>The value of variable_name, bad_value, is not a valid
integer. (err-misc)
```

Specifying Boolean values

When an environment variable has a Boolean value, you can specify any nonnull value except `0` to set the variable. When the value for a Boolean variable is `false`, it means that the variable is not set. To turn off a Boolean variable, set it to `0`.

A `true` setting for an environment variable is anything nonempty or nonnull that does not completely parse as a constant to the value of `0` according to the rules of the C programming language. For example, the following values return `true`:

- `0Foo` and `Foo`
- `23`
- `0 0`
- <space>`1`
- `false`, `False`, and `FALSE`

The following values return `false`:

- `0`
- `00`
- <space>`0`
- `0` multiplied by `0`

OS_16K_PAGE

Default: `false`

When `true` (nonzero), the `OS_16K_PAGE` variable specifies a page size of 16384 bytes to be used by ObjectStore. The default size is defined by the system. Note that you cannot use this variable to set the page size smaller than the system-defined size.

Caution — Use the `OS_16K_PAGE` environment variable carefully. If you think you need to use it, contact Technical Support.

OS_32K_PAGE

Default: `false`

When `true` (nonzero), the `OS_32K_PAGE` variable specifies a page size of 32768 bytes to be used by ObjectStore. The default size is defined by the system. Note that you cannot use this variable to set the page size smaller than the system-defined size.

Caution — Use the `OS_32K_PAGE` environment variable carefully. If you think you need to use it, contact Technical Support.

OS_64K_PAGE

Default: `false`

When `true` (nonzero), the `OS_64K_PAGE` variable specifies a page size of 65536 bytes to be used by ObjectStore . The default size is defined by the system. Note that you cannot use this variable to set the page size smaller than the system-defined size.

Caution — Use the `OS_64K_PAGE` environment variable carefully. If you think you need to use it, contact Technical Support.

OS_8K_PAGE

Default: `false`

When `true` (nonzero), the `OS_8K_PAGE` variable specifies a page size of 8192 bytes to be used by ObjectStore. The default size is defined by the system. Note that you cannot use this variable to set the page size smaller than the system-defined size.

Caution — Use the `OS_8K_PAGE` environment variable carefully. If you think you need to use it, contact Technical Support.

OS_ALWAYS_CHECK_SERVER_AT_COMMIT

Default: Not set

The OS_ALWAYS_CHECK_SERVER_AT_COMMIT variable specifies whether ObjectStore applications check for valid server connections before committing read-only transactions.

To set this variable, specify any nonnull value except 0.

By default, this variable is not set and ObjectStore applications do not perform this check.

See also objectstore::set_always_check_server_connection_at_commit() in the *C++ API Reference*.

OS_AS_SIZE

Default: by platform

The OS_AS_SIZE variable sets the size, in bytes, of the persistent storage region in the address space for each client. Specify values in numbers of bytes that are 0x10000 aligned (64 KB). Typically, it is not necessary to specify a value for this variable. The default value varies from platform to platform, as shown in the table below.

Platform	***Default Size OS_AS_SIZE***
AIX	1024 MB (0x40000000)
Intel Windows	512 MB (0x20000000)
Linux 32 bit	255 MB (0xFFF0000)
Linux 64 bit	
Solaris 32 bit	192 MB (0xC000000)
Solaris 64 bit	192 MB (0xC000000)

The size of the persistent storage region on the creating platform typically determines the size of the largest object you can store in a database. This is because ObjectStore commits the entire object in the transaction that allocates it. However, if the maximum size of a cluster is smaller than OS_AS_SIZE, the largest object you can store is limited to the maximum cluster size. (Allocation is limited to a single cluster.)

Caution

An incorrect value for OS_AS_SIZE or OS_AS_START can cause failures. Be absolutely certain that you understand the way addresses are assigned on your platform before you modify these values.

For example, if you quadruple OS_AS_SIZE, your application can run on a Solaris 32 bit, but fails on Windows systems.

Address space considerations

When considering address space, note the important distinction between assigning an address and mapping an address. When ObjectStore assigns an address, it means that the client determines where it would put those pages if it needed them. Assigning an address reserves space so that the address cannot be assigned to another page. When ObjectStore maps a page to an address, it means that the page is available to the client.

When a page uses hard pointers to refer to pages outside the region, ObjectStore assigns address space to those pages. This makes it possible, in certain situations, to use up available address space by touching a single page. In many cases, changing `OS_AS_SIZE` solves this problem. However, there are cases in which the schema design could benefit from the strategic application of soft pointers and sometimes a reworking of transaction semantics.

If an application uses soft pointers instead of hard pointers to point to data in other regions, ObjectStore does not assign address space to the other regions until the soft pointers are actually resolved. In programs with complex schemas using pages that refer to objects in many other regions (a high degree of "nonlocal fan out"), use soft pointers instead of hard pointers.

If the client exhausts the address space, it might be possible to reorganize some of the client's transactions to reduce the amount of data being referenced in any single transaction.

OS_AS_START

Default: not set

The `OS_AS_START` variable sets the address of the beginning of the persistent storage region (PSR) of the process's address space. Specify values that are 0x10000 aligned (64 KB). Typically, you do not supply a value for this variable. The default is that this variable is not set and the operating system chooses a location for the PSR.

See `OS_AS_SIZE` on page 103 for a caution about the use of this variable.

Windows

On Windows, 2 GB is the maximum user-mode address space. Your program, its data, and ObjectStore's address space must fit in 2 GB. A user program on Windows uses the bottom 2 GB of address space. DLLs and the like use up space at 0x10000000; stacks use up space at 0x7fxxxxxx. Consequently, there are only 1.75 GB available.

Linux

If you use the `OS_AS_START` environment variable on Linux platforms, you are required to also set the `OS_USE_MAP_FIXED` environment variable. See OS_USE_MAP_FIXED on page 128 for more information.

Solaris

If you use the `OS_AS_START` environment variable on Solaris platforms and the operating system reports that the specified address region is unavailable, but you know by other diagnostic methods that the region really is available, you can set the `OS_USE_MAP_FIXED` environment variable to override the operating system's message. See OS_USE_MAP_FIXED on page 128 for more information.

OS_ASMARKERS_USELESS

Default: `false`

The `OS_ASMARKERS_USELESS` variable allows you to disable address-space markers. By default, the markers are enabled.

You can also control and retrieve the setting of address-space markers programmatically; see `objectstore::get_asmarkers_useless()` and `objectstore::set_asmarkers_useless()` in the *C++ API Reference*.

OS_AUTH

The OS_AUTH variable overrides the authentication search protocol (see Authentication Required on page 80). The format for specifying this variable is OS_AUTH=*type* or OS_AUTH=*host*:*type*, where *type* is one of the values listed in the table below and *host* is the name of the server.

The valid types that OS_AUTH can be assigned are as follows:

Value	*Authentication*
NONE	No authentication
SYS	Sun ONC RPC AUTH_SYS authentication
DES	ONC RPC AUTH_DES authentication
NP	UNIX seteuid/setegid or Windows LogonUser authentication (Name Password)
NT_LOCAL	Windows NT ImpersonateLoggedOnUser authentication
NT_REMOTE	Windows NT Security Service Provider authentication

You can also use a semicolon-separated list of authentication types to specify authentication types for various hosts. This list can include one default value. See examples below.

OS_AUTH can specify only one default authentication type, and it can specify only one authentication type per host. The client performs network name lookup, so it is not necessary to list the aliases of a host. For example, if the default DNS domain is odi.com and a server has two network names — server1 and server-one — the host names server1, server-one, and server1.odi.com are all the same host.

If OS_AUTH specifies a host-specific authentication type that is not supported by the server, the default value is used if the server supports the default.

If the server supports neither the specified authentication type nor the default type, the client attempts to find a type supported by both the client and the server. If it does not find a supported authentication type, it signals an err_authentication_failure error.

Examples

```
OS_AUTH=server1:DES;SYS
```

Instructs the client to use SYS authentication for all servers except server1. When connecting to server1, the client uses DES authentication if server1 supports it. If server1 does not support DES or SYS, an authentication method is selected from the list of authentication types that the server supports.

```
OS_AUTH=NONE
```

Instructs the client to use no authorization. If a server does not support the NONE type, an authentication method is selected from the list of authentication types the server supports.

```
OS_AUTH=server1:NP
```

Instructs the client to use Name Password authentication when connecting to `server1`. For any other server, an authentication method is selected from the list of authentication types that the other server supports.

OS_CACHE_DIR (UNIX Only)

Default: `/tmp/ostore`

The `OS_CACHE_DIR` variable specifies the path of the directory used for the client cache and communications segment (commseg). If `/tmp` is very small, it might be desirable to locate the cache file and commseg elsewhere.

ObjectStore places the cache file and commseg in the specified directory and assigns a file name to avoid conflicts between multiple processes that are running ObjectStore and using the same directories.

If `OS_CACHE_DIR` is not set, ObjectStore places the cache file in the directory specified by the `Cache Directory` parameter in the cache manager parameters file, if one exists. See Setting Cache Manager Parameters on page 307.

Caution

This directory should not be an NFS mount point; if it is, the result can be slower client performance and potential problems with memory mapping over NFS.

Cache and commseg files should be in the same directory. See `OS_COMMSEG_DIR (UNIX Only)` on page 108 for related information.

Windows

The operating system determines the location of the cache in virtual memory. You cannot change the location.

OS_CACHE_SIZE

Default: 8 MB

The `OS_CACHE_SIZE` variable specifies the size of the client cache, in bytes. The default cache size can be overridden using the `OS_CACHE_SIZE` environment variable.

Note that the size of the client cache does not limit the size of an object that can be in the database. ObjectStore can store an object on multiple pages and can swap pages in and out of the client cache as needed.

When you are trying to determine the optimum cache size for your application, consider data access patterns as well as the amount of data that is accessed. In other words, both size and operation are important. The goal is to minimize the number of times the client must swap pages out of the cache and send them back to the server.

Also consider the amount of physical memory in your machine. Usually, it is desirable for the cache to stay in physical memory rather than to be swapped out to disk.

The cache size is limited only by the amount of resources (address space, memory, or disk space) on the machine.

If necessary, ObjectStore rounds down the number you specify to a multiple of the page size.

OS_CMGR_OUTPUT_LOG_NAME

Default: not set

This environment variable affects the cache manager and the client that starts the cache manager.

The `OS_CMGR_OUTPUT_LOG_NAME` variable specifies the name of the cache manager log file to use instead of the default name. If this variable is not set, the log file is assigned the default name of `osc6_out` on UNIX or `oscmgr6.txt` on Windows in the `temp` directory.

OS_CMGR_STARTUP_LOCK (UNIX Only)

Default: by platform

The `OS_CMGR_STARTUP_LOCK` variable specifies an alternative location for the cache manager startup lock file.

ObjectStore clients automatically start a cache manager if one is not already running on UNIX systems. ObjectStore uses a special file, called a *startup lock file*, to ensure that only one cache manager starts on a host. The default location for this file is according to platform:

- Solaris 2: `/var/tmp/cmgr6_startup_lock`
- Other systems (not Windows): `/tmp/cmgr6_startup_lock`

If a cache manager is not already running, a client

1. Creates the startup lock file
2. Launches the cache manager
3. Ensures that the client can communicate with the cache manager
4. Deletes the startup lock file

The startup lock file does not contain anything. Its existence serves as a lock while a client is starting a cache manager, which prevents other clients from also starting a cache manager.

If you do not want to use the default location for the startup lock file, or if the default location is unavailable, you can specify an alternative location with the `OS_CMGR_STARTUP_LOCK` environment variable. Set this variable to a local path name.

ObjectStore tries to use the default location first. If it is not possible to use the default, ObjectStore uses the path name you specified for `OS_CMGR_STARTUP_LOCK`.

The client process must have the authority to create the specified file.

OS_COLL_DEBUG_INDEX

Default: Not set

The `OS_COLL_DEBUG_INDEX` variable is used to debug problems that may occur when an application adds indexes to a collection or inserts elements into an index.

When set, this variable instructs ObjectStore to check that

- New indexes added to a collection will expect the same type of element as do previously added indexes for that collection
- New indexes added to a collection match the type of existing elements in the collection
- The type of elements inserted into an index match the type of elements in existing indexes on the collection

If this variable is set, running the utility `osverifydb` will check the consistency of the type of all indexes on each collection with the type of each member of that collection. See osverifydb on page 258 for more information.

Example

Here is a sample of the displayed output from index operations when this variable is set:

```
DBG: Adding index of type "A*" to collection at 0xe14b0010
DBG: Checking element at 0xe14b08d4 of type "A" OK
DBG: Checking element at 0xe14b08e8 of type "A2" OK ("A" is a
   base of "A2")
DBG: Checking element at 0xe14b08d8 of type "A" OK
DBG: Checking element at 0xe14b08dc of type "B" INCONSISTENCY
```

OS_COLLECTION_TRACE_INDEX_USAGE

Default: Not set.

When set, the `OS_COLLECTION_TRACE_INDEX_USAGE` variable sends index use informtion to `stdout` after executionof each query in your program. For information about how to use this variable, see *C++ Collections Guide and Reference*, Indexes and Query Optimization, Monitoring Index Use During Queries.

OS_COMMSEG_DIR (UNIX Only)

Default: `/tmp/ostore`

The `OS_COMMSEG_DIR` variable specifies the path of the directory used for the communications segment (commseg). ObjectStore places the commseg in the `/tmp/ostore` directory automatically. If the UNIX file system containing `/tmp/ostore` is very small, it might be desirable to locate the commseg elsewhere.

ObjectStore places the commseg in the specified directory and assigns a unique file name to avoid conflicts between multiple processes that are all running ObjectStore and using the same `OS_COMMSEG_DIR` setting.

Commseg and cache files should be in the same directory. See `OS_CACHE_DIR (UNIX Only)` on page 106 for related information.

If `OS_COMMSEG_DIR` is not set, ObjectStore places the commseg in the directory specified by the `Commseg Directory` parameter in the cache manager parameters file, if one exists. See Setting Cache Manager Parameters on page 307.

Caution

This directory should not be an NFS mount point because this can result in slower client performance and can create potential problems with memory mapping over NFS.

Windows — The operating system determines the location of the commseg in shared memory. You cannot change the location.

OS_COMP_SCHEMA_CHANGE_ACTION

Default: `error`

The `OS_COMP_SCHEMA_CHANGE_ACTION` variable controls the severity of the error resulting from a type mismatch during library and compilation schema generation. You can specify the following values:

Value	*Description*
`error`	The type mismatch is reported as an error. The compilation is eventually terminated and the compilation schema remains unchanged.
`silent`	The type mismatch is not reported. The new type definition replaces the previous definition in the compilation schema.
`warn`	The type mismatch is reported as a warning. The new type definition replaces the previous definition in the compilation schema.

OS_CORE_DIR (UNIX Only)

Default: root directory

This environment variable affects the server, the cache manager, and the client that starts the cache manager.

The `OS_CORE_DIR` variable specifies the location where the server and cache manager are to dump core files.

OS_DEBUG_C0000005 (Windows Only)

Default: `false`

The `OS_DEBUG_C0000005` variable instructs ObjectStore to display a message box if a fault occurs and the ObjectStore exception handler sees it. The message explains the fault as an `Attempt to read location 0x1234 from EIP 0x10231023.` This can be useful in locating access violations in your code. To set this variable, specify any nonnull value except `0`.

OS_DEBUG_LOCATOR_FILE

Default: `0`

The `OS_DEBUG_LOCATOR_FILE` variable instructs ObjectStore to send diagnostic information about the processing of the locator file to `stderr`.

When this variable is set to `0`, the default, no information is provided.

When this variable is set to `1`, ObjectStore generates a report every time the locator file is searched. This allows you to determine the input to the search exactly, which

in turn enables you to diagnose the reason that the locator file is not providing the expected results.

If your application is a Windows GUI application, see OS_STDOUT_FILE (Windows Only) on page 125.

OS_DEF_BREAK_ACTION (Windows Only)

Default: `false`

This environment variable affects the server, client, and cache manager.

The `OS_DEF_BREAK_ACTION` variable allows you to set a breakpoint that obtains a stack trace before the stack is unwound. When you set this variable to 1, ObjectStore reaches a hard-coded breakpoint immediately before an exception is signaled. This is useful when your application exits with an unhandled TIX exception and works with Visual C++'s just-in-time debugging. Setting `OS_DEF_BREAK_ACTION` also hits a breakpoint if ObjectStore's internal abort routine is called.

OS_DEF_EXCEPT_ACTION

Default: Not set

This environment variable affects the server, client, and cache manager.

The `OS_DEF_EXCEPT_ACTION` variable controls what happens if an unhandled exception is signaled. It is useful in debugging unhandled exceptions. You can specify the following values:

Value	*Description*
`abort`	ObjectStore aborts the process. On UNIX, ObjectStore creates a core file and, if you are running in a debugger, returns control to the debugger.
`integer`	Specify an integer greater than or equal to 1. ObjectStore exits from the program with the specified integer as the return value.
`kill`	ObjectStore action varies by platform: • Windows — `ExitProcess (0x006600);` • UNIX — `kill (getpid(), SIGKILL);`
Any other value	ObjectStore exits from the program with a return value of `1`. This is the default.

OS_DEF_MESSAGE_ACTION (Windows Only)

Default: Not set

This environment variable affects the server, client, and cache manager.

The `OS_DEF_MESSAGE_ACTION` environment variable determines the default message action for `_ODI_message` (used for unhandled TIX exceptions). You can set this variable to `stderr` or `stdout` to send the information there, or you can specify the name of a file to receive the information.

OS_DEFAULT_AS_PARTITION_SIZE

Default: Not set

The `OS_DEFAULT_AS_PARTITION_SIZE` variable specifies, in bytes, the amount of address spaced assigned to newly created sessions unless overridden by `objectstore::set_default_address_space_partition_size()`. The actual amount assigned is the specified amount rounded up to the next multiple of 64 KB.

For more information on sessions, see Chapter 3, Multithread and Multisession Applications, in the *Advanced C++ API User Guide*.

OS_DISABLE_PROPAGATE_ON_COMMIT

Default: `false`

Intended for use with ObjectStore/Single, `OS_DISABLE_PROPAGATE_ON_COMMIT` controls when ObjectStore propagates committed data from the transaction log to a database. By default, this environment variable is `false` and data is propagated to a database when a top-level transaction commits. If this environment variable is set to `true`, the data is propagated when a database is closed and when an application ends normally. If an application ends prematurely, some of the data might not have been propagated to the database. This data is propagated the next time the ObjectStore is initialized or on a call to `progagate_log()`.

For some applications, setting this environment variable to `true` can improve performance because data is written to the database less frequently. However, doing so increases the risk of data loss, which can happen if an application ends prematurely and the transaction log is deleted before the data is propagated to the database.

Applications can override this environment variable by using `objectstore::set_propagate_on_commit()`. For more information, see `objectstore::get_propagate_on_commit()` and `objectstore::set_propagate_on_commit()` in the *C++ API Reference* and Chapter 6, Working with ObjectStore / Single, in *Building C++ Interface Applications*.

OS_DISALLOW_OSSINGLE_REMOTE_DATABASES

Default: Not set

This variable is for use with ObjectStore/Single. When the `OS_DISALLOW_OSSINGLE_REMOTE_DATABASES` environment variable is set, ObjectStore/Single applications are prevented from accessing databases on remote file systems. To set this variable, specify any nonnull value except `0`.

By default, this variable is not set and ObjectStore/Single applications can access databases on remote file systems.

If this environment variable is set and the application detects a remote database during `os_database::open()` or `os_database::create()` the exception `err_file_not_local` is raised.

For more information, see Chapter 6, Working with ObjectStore / Single, in *Building C++ Interface Applications*.

OS_DISPLAY_INSTALL_MISMATCHES

Default: `false`

The `OS_DISPLAY_INSTALL_MISMATCHES` variable displays to `stdout` any mismatches found during Metaobject Protocol (MOP) dynamic type installation. When this variable is not set, the only indication of failure is that an exception is raised. To set this variable, specify any nonnull value except `0`.

If your application is a Windows GUI application, see OS_STDOUT_FILE (Windows Only) on page 125.

OS_ENABLE_DECACHE_SOFT_POINTERS_AFTER_AS_RELEASE

Default: `false`

The `OS_ENABLE_DECACHE_SOFT_POINTERS_AFTER_AS_RELEASE` variable enables automatic soft-pointer decaching after any address space is released. By default, ObjectStore does not automatically decache soft pointers on a page when that page is accessed again. When a page that contains cached soft pointers is reused, the address space for the soft-pointer targets is immediately reserved. When soft-pointer decaching is enabled and a page containing cached soft pointers is reused, the soft pointers on the page are decached, ensuring that address space is not reserved for their targets.

To enable soft-pointer decaching, set this variable to `true` (any nonnull value except `0`). For more information about soft-pointer decaching, see Soft-Pointer Decaching in Chapter 1 of the *Advanced C++ API User Guide*.

OS_ENABLE_REALTIME_COUNTERS

Default: `false`

The `OS_ENABLE_REALTIME_COUNTERS` variable turns on the collection and display of timing statistics that measure ObjectStore performance. ObjectStore displays values of real-time counters that help show where time is being spent. To set this variable, specify any nonnull value except `0`.

OS_FORCE_HANDLE_TRANS

OS_HANDLE_TRANS (UNIX Only)

Default: `false`

The `OS_FORCE_HANDLE_TRANS` and `OS_HANDLE_TRANS` variables control what ObjectStore does if there is a memory fault on an address that is not in the persistent storage region of address space; for example, if you happen to dereference a null pointer.

When you set `OS_FORCE_HANDLE_TRANS` to any nonnull value except `0` (or `objectstore::set_handle_transient_faults()` is called with the `true` argument) ObjectStore signals the appropriate exception: `err_null_pointer` or `err_deref_transient_pointer`. This causes dereferences to illegal non-ObjectStore addresses to signal a TIX exception and display a message, and lets you get a stack trace.

When you set `OS_HANDLE_TRANS`, ObjectStore runs the `SIGSEGV` handler that was in force before ObjectStore was fully initialized. If there is no such handler, ObjectStore signals the appropriate exception.

If neither of these variables is set, ObjectStore runs the `SIGSEGV` handler that was in force before ObjectStore was fully initialized. If there is no such handler, ObjectStore

returns the signal to the operating system to handle. The operating system performs the usual actions for SIGSEGVs, for example, dumping core or some other action that depends on other environment variable settings.

If you do not set up your own SIGSEGV handler before ObjectStore is initialized, the error message is Segmentation Violation: Core Dumped. If both variables are set, OS_FORCE_HANDLE_TRANS has precedence.

OS_IGNORE_LOCATOR_FILE

Default: false

The OS_IGNORE_LOCATOR_FILE variable indicates that no locator file is associated with any application on the client. This overrides all other settings and function calls, including the existence of $OS_ROOTDIR/etc/locator. Specify any nonnull value except 0 to set this variable. Set this variable to 0 to allow locator files (this is the default).

When this variable is set to a nonnull nonzero value, locator files are not used. However, a small part of the locator file logic in the client is executed nonetheless. Consequently, if you have locator file debugging enabled, you still receive diagnostic information. See OS_DEBUG_LOCATOR_FILE on page 109.

For information about the locator file, see Description of the Locator File on page 267.

OS_INC_SCHEMA_INSTALLATION

Default: false

When the OS_INC_SCHEMA_INSTALLATION variable is set to any nonnull value except 0, ObjectStore creates new databases in incremental schema installation mode. This means that ObjectStore adds types to the database schema as they are needed. When this variable is not set, ObjectStore creates new databases in batch schema installation mode. This means that when ObjectStore creates a database, it creates a database schema that includes all types that might be allocated in the database.

An application performs incremental rather than batch schema installation if either of these conditions is true:

- The application has OS_INC_SCHEMA_INSTALLATION set to any nonnull value except 0.
- The database was created with incremental schema installation.

You can override the effect of this environment variable for a particular process by using objectstore::set_incremental_schema_installation(). See the *C++ API Reference*. For a discussion of the issues involved, see the *C++ API User Guide*.

OS_INHIBIT_TIX_HANDLE

Default: not set

This environment variable affects the server, client, and cache manager.

The `OS_INHIBIT_TIX_HANDLE` variable specifies an error message substring for which exception handling is to be disabled.

Many end-user applications have omnibus error handlers to catch all errors being signaled and to present them in an easily readable format to the user. This sometimes makes debugging difficult because the backtrace information has disappeared. When you specify a substring to `OS_INHIBIT_TIX_HANDLE`, if the substring appears in the formatted error message, exception handling is disabled for the specific error. You can then generate an unhandled exception dump for analysis or view the backtrace in a debugger.

Specify `all` to disable all handling.

OS_LANG_OVERRIDE (UNIX Only)

Default: not set

This environment variable affects the server, client, and cache manager.

The `OS_LANG_OVERRIDE` variable changes the behavior of ObjectStore without affecting other applications that depend on `LANG`.

`LANG` is a public environment variable that controls the selection of the message catalog sets that are appropriate for the specific language locale. It also is used internally to identify the character encoding used and, thus, affects the processing of strings. This variable might already be set by your system manager.

Because the `LANG` variable is public, ObjectStore supplies another variable, `OS_LANG_OVERRIDE`, that takes precedence over `LANG`. The result is that anywhere ObjectStore uses catalogs, `OS_LANG_OVERRIDE` can change the catalog path construction as well as the way strings are processed. `OS_LANG_OVERRIDE` also affects `.msg` file path construction for ObjectStore scripts such as `osinstal` and `osconfig` on platforms that use them.

There is a possibility that the value of `LANG` is not one of the variables shown in the following table. If the value set is not listed in the table, English is assumed unless `OS_LANG_OVERRIDE` is set to one of the values in the table.

The values recognized by ObjectStore appear in the following table:

Value	*HP–UX*	*Non-HP–UX*
`japanese`	Shift JIS Code (SJIS)	Extended UNIX Code (EUC)

`Ja_JP`
`jp_JP.sjis`
`ja_JP.sjis`
`japanese.sjis`
`ja__JP.PCK`
`jp__JP.PCK`
	Shift JIS Code (SJIS)	Shift JIS Code (SJIS)

`Japanese`
`japanese.euc`
`japan`
`Japan`
`ja`
`jp_JP`
`jp_JP.euc`
`ja_JP`
`ja_JP.euc`
	Extended UNIX Code (EUC)	Extended UNIX Code (EUC)

OS_LIBDIR

Default: `OS_ROOTDIR\lib`

The `OS_LIBDIR` variable specifies the directory that contains schema files distributed with ObjectStore. The directory must be on a machine that has a server, so you do not receive an error when trying to access the schema files. This is set up by the installation process. The only exception to this is if you have a locator file set up to handle remote access to databases and you configure the relevant server to allow remote access to databases.

If `OS_ROOTDIR` is on a machine that does not have an ObjectStore server, you must copy or move the library schema files to a machine that is running a server. Use this variable to specify the new directory.

OS_LOCATOR_ESCAPE_CHARACTER

Default: `$`

The `OS_LOCATOR_ESCAPE_CHARACTER` variable specifies the escape character for regular expressions in an ObjectStore locator file. The default escape character for regular expressions in locator files is a dollar sign (`$`), rather than the usual backslash (\). You can specify an escape character explicitly with this environment variable. For information about the locator file, see Description of the Locator File on page 267.

OS_LOCATOR_FILE

Default: `false`

The client environment variable `OS_LOCATOR_FILE` can be set to any legitimate argument to `objectstore::set_locator_file()`, with the same meaning. Calls to `set_locator_file()` override this setting. For more information about `set_`

`locator_file()` see Chapter 2, Class Library, in the *C++ API Reference*. See also Description of the Locator File on page 267 of this book.

OS_LOG_TIX_FORMAT

Default: `false`

This environment variable affects the server, client, and cache manager.

The `OS_LOG_TIX_FORMAT` variable specifies the name of a log file to record all exceptions signaled. This file logs all `printf` control strings signaled, regardless of whether the exception is handled. This facility is especially useful for debugging two situations: recursive exceptions (common if you get exceptions during message processing) and bad `printf` strings.

OS_META_SCHEMA_DB

Default: `$OS_ROOTDIR/lib/metaschm.db`

The `OS_META_SCHEMA_DB` variable specifies the location of the metaschema database.

The metaschema database, which is shipped with ObjectStore, describes hidden internal types and is needed for operations such as data browsing, schema evolution, database verification, and the Metaobject Protocol (MOP).

The metaschema database must be on the same machine as the server. The only exception to this is if you have a locator file that is set up to handle remote access to databases and you configure the relevant server to allow remote access to databases.

Normally, the metaschema database is in `$OS_ROOTDIR/lib`, but under certain circumstances, it might be elsewhere. This can happen when the machine on which `$OS_ROOTDIR` resides does not have a local ObjectStore server. In this situation, unless you set up a locator file, you must copy the metaschema database to the machine that the server runs on.

If you move the metaschema database out of `$OS_ROOTDIR/lib`, be sure to set this variable to the new location so that ObjectStore can find the metaschema database.

OS_NETWORK (Windows Only)

Default: `O6NETNSM, O6NETTCP`

This environment variable affects the server, client, and cache manager.

The `OS_NETWORK` variable specifies the network dynamic link libraries (DLLs) to be loaded and used by all ObjectStore executables.

Normally, you need not change the default setting. In other words, you do not usually set `OS_NETWORK`. Possible values for `OS_NETWORK` are

- `O6NETTCP` — TCP/IP network DLL. Used for network communication.
- `D6NETTCP` — TCP/IP network DLL. Used for network communication. Use for ObjectStore DEBUG builds.

- O6NETNSM — Named shared memory DLL. Used for local interprocess communication.
- D6NETNSM — Named shared memory DLL. Used for local interprocess communication. Use for ObjectStore DEBUG builds.

In standard operation, ObjectStore automatically detects the networks that are present, and OS_NETWORK does not need to be set. However, you can set OS_NETWORK to a comma-delimited list in situations in which you want more control over the network used by ObjectStore.

The order of the list is important. When you set the OS_NETWORK variable, you must almost always specify O6NETNSM, the local network for Windows, as the first network in the list. For local connections, this network offers significantly higher performance than any other network.

If you omit the local network or do not specify it first, starting an ObjectStore application might fail to start the cache manager automatically. You must then start the cache manager manually before running the first ObjectStore application. Many other operations can fail if these DLLs are not specified at all or are not specified first.

When to change the default

Suppose that you originally installed ObjectStore on a system connected to a network and then you disconnect the system from the network. You must set OS_NETWORK so that it no longer expects to enable network protocols. For example, if your system was previously using TCP and now it is a stand-alone system, you might receive the following error when you try to run ObjectStore:

```
ObjectStore internal error
connect failed (err_internal)
```

Set OS_NETWORK to O6NETNSM on Windows to prevent the TCP interface from loading and to allow the server to initialize normally.

On Windows, if you have an unsupported TCP/IP stack on your machine and you experience problems, set OS_NETWORK to O6NETNSM so that ObjectStore does not attempt to initialize or use the unsupported network.

OS_NETWORK_SERVICE (UNIX Only)

Default: echo

This environment variable affects only the server.

The OS_NETWORK_SERVICE variable specifies which of the Internet services listed in /etc/services should be used for ObjectStore networking. The default Internet service used by ObjectStore is echo. If echo is not available when starting up the ObjectStore server, you can set this variable should be set to another service.

The OS_NETWORK_SERVICE variable applies only to UNIX SVR4 and has no effect on other operating systems.

OS_NO_MAPPED (UNIX Only)

Default: `true`

When the `OS_NO_MAPPED` variable is set to `false` (`0`), ObjectStore does not use mapped communication between the client and the server. Note that ObjectStore uses mapped communication only if the client and the server are on the same host and the server parameter `Allow Shared Communications` is not set to `no`; see Allow Shared Communications (UNIX Only) on page 80. To set the `OS_NO_MAPPED` variable to `true` (the default), specify any nonnull value except `0`.

OS_NOTIFICATION_QUEUE_SIZE

Default: 50

The `OS_NOTIFICATION_QUEUE_SIZE` variable specifies the maximum number of notifications that can be in a process's notification queue. Notification queues are part of the ObjectStore cache manager process. The cache manager has a notification queue for each local client.

If an application does not call the `set_queue_size()` function, ObjectStore uses the value you specify for `OS_NOTIFICATION_QUEUE_SIZE`. If an application does not call `set_queue_size()` and this environment variable is not set, ObjectStore uses a default value of `50`.

The maximum allowable value for this variable is `16384`. If you set this variable to a higher value, ObjectStore uses `16384` as the value, rather than the value you set.

The function for setting the maximum length of the notification queue is `os_notification::set_queue_size()`.

OS_OSDUMP_APPSCHEMA_PATH

The `OS_OSDUMP_APPSCHEMA_PATH` variable allows users to provide a different path for `osdump.adb`.

OS_OSLOAD_APPSCHEMA_PATH

The `OS_OSLOAD_APPSCHEMA_PATH` variable allows users to provide a different path for `osload.adb`.

OS_OSSG_CPP

Default: by platform

The `OS_OSSG_CPP` variable sets the C preprocessor used by `ossg`. You can set this variable to a nondefault C preprocessor. On UNIX, the default preprocessor is `cpp` and on Windows, it is `cl`.

OS_OSVERIFYDB_BUFFER_SIZE

Default: 512 MB

The `OS_OSVERIFYDB_BUFFER_SIZE` variable specifies the maximum size of the transient buffer used to optimize locality of reference by the `osverifydb` utility in fast mode.

OS_OSVERIFYDB_FAST

Default: false

The `OS_OSVERIFYDB_FAST` variable specifies that the `osverifydb` utility will run in fast mode. This is equivalent to running the `osverifydb` utility with the `-F` option.

OS_PASSWORD

The `OS_PASSWORD` variable specifies a password to be used by a client application to connect to an ObjectStore server, used with the `OS_USERNAME` environment variable. If the client allows `Name Password` authentication, the client connects to the server without prompting for a username and password.

OS_PATHNAME_ENCODING

Default: by platform

This environment variable affects the server, client, and cache manager. For ObjectStore client applications written in the Java interface, this environment variable has no effect.

ObjectStore supports multibyte encoded path names. The `OS_PATHNAME_ENCODING` variable specifies a character set encoding to override the default encoding. The default encoding is determined by the system on which the client application is running. The valid values that can be specified for this variable are:

- `ASCII` – 7 bit ASCII.
- `CP1252` – Microsoft Code Page 1252 (US English).
- `CP932` – Microsoft Code Page 932 (Japanese).
- `EUCJP` – Extended UNIX Code (Japanese).
- `UTF8` – UCS Transformation Format 8.
- `NONE` – No encoding translation; values 0x01 through 0xFF are passed through without modification.

OS_PORT_FILE

Default: by platform

This environment variable affects the server, client, and cache manager.

The `OS_PORT_FILE` specifies the name of a ports file for network services. See Modifying Network Port Settings on page 58.

OS_PREALLOCATE_CACHE_FILES (UNIX Only)

Default: `true`

The `OS_PREALLOCATE_CACHE_FILES` environment variable controls the preallocation of cache and commseg files.

Note

This environment variable is for use with ObjectStore / Single applications only. To control preallocation on other ObjectStore applications running on UNIX platforms, use the `Preallocate Cache Files` cache manager parameter, as described in Preallocate Cache Files Parameter on page 310.

By default, ObjectStore preallocates cache files during client initialization. If the files are large (hundreds of megabytes or more), preallocation can cause a noticeable delay during initialization. To increase performance, set this parameter to `false` (`0`), preventing the up-front allocation of disk blocks to cache and commseg files. Instead, space is allocated for the files as needed. Setting this environment variable to `true` (nonzero) restores the default behavior.

Caution

When running clients with the `OS_PREALLOCATE_CACHE_FILES` environment variable set to `false`, you must ensure that the file system that contains the cache and commseg files never gets full. If the system is unable to allocate free space for the cache files, the client or server can crash.

OS_PRINT_CLIENT_COUNTERS

Default: `false`

The `OS_PRINT_CLIENT_COUNTERS` variable turns on display of counters that provide information about client performance. To set this variable, specify any nonnull value except `0`.

OS_RCVBUF_SIZE

Default: 16384 bytes

The `OS_RCVBUF_SIZE` variable sets the default size of the network buffer used by the client to receive data from the server. For best results, this size should be the same as that specified by the server parameter `Preferred Network Send Buffer Size`. See `Preferred Network Send Buffer Size` on page 95 for further information.

Note that some applications benefit from an increase in the size of the network buffers used by ObjectStore clients and servers. You can change the size used by the clients from the default of 16384 bytes by setting the environment variables `OS_SNDBUF_SIZE` and `OS_RCVBUF_SIZE`. You can change the size used by the server by setting the server parameters `Preferred Network Send Buffer Size` and `Preferred Network Receive Buffer Size`. Usually, you achieve the best performance if `OS_SNDBUF_SIZE` is the same as `Preferred Network Receive Buffer Size` and `OS_RCVBUF_SIZE` is the same as `Preferred Network Send Buffer Size`.

Depending on the operating system, you might find that large values for this variable are rejected, which leads to reduced performance. ObjectStore Technical

Support recommends that you experiment by doubling the size until performance no longer improves.

OS_REMOTE_AUTH_REGISTRY_LOCATION (Windows Only)

Default: `Object Design Inc.\ObjectStore 6.0`

The `OS_REMOTE_AUTH_REGISTRY_LOCATION` variable sets the Windows registry location for ObjectStore clients. It specifies a location relative to `HKEY_LOCAL_MACHINE\SOFTWARE`. For example, if the environment variable is set to `my_location`, the registry location is

```
HKEY_LOCAL_MACHINE\SOFTWARE\my_location
```

You can also set the registry location programmatically by calling `os_authentication::set_nt_registry_location()`. For more information, see `os_authentication::set_nt_registry_location()` in the *C++ API Reference*. For information about setting the registry location, see Changing the Registry Location for ObjectStore (Windows Only) on page 325.

OS_RESERVE_AS (UNIX Only)

Default: `false`

The `OS_RESERVE_AS` variable specifies whether ObjectStore's reserve address space mode is on or off. When off, ObjectStore is optimized to increase performance, sometimes by a very significant factor. ObjectStore uses this optimization when this environment variable is not set. This means that performance is fast but that there is potential for confusion. ObjectStore does not use the optimization when this environment variable is set.

This optimization can cause problems if your own program calls the `mmap` system call with `0` as the first argument or if your program calls a subroutine library that does so. If your program does either of these things, you should disable the optimization either by calling the entry point `objectstore::set_reserve_as_mode(os_boolean new_mode)` or by setting the environment variable `OS_RESERVE_AS` to any nonnull value except `0`. If you both call the entry point and set `OS_RESERVE_AS`, the entry point takes precedence.

This variable affects the setting of reserve address space mode. When ObjectStore is in this mode, it always keeps the entire persistent region reserved, from the operating system's point of view, so that any other subsystem in the client process that maps something in and asks the operating system to assign some address space receives address space outside the persistent region.

If reserve address space mode is off, such a request might assign space that is part of ObjectStore's persistent storage region. This can cause problems because the subsystem appears unable to coexist with ObjectStore.

If reserve address space mode is off, however, operating system calls that manipulate the virtual address space are faster on some platforms.

OS_ROOTDIR

Default: by platform

This environment variable affects the server, client, and cache manager.

The OS_ROOTDIR variable specifies the top-level directory in the part of the file system hierarchy containing ObjectStore files. The name specified by this variable serves as the prefix of various directory names used in search paths. This environment variable is required to run ObjectStore. You set the value of the variable when you install ObjectStore.

If you change the location of your ObjectStore installation, be sure to change the value specified for OS_ROOTDIR. Not doing so can cause the following message to be signaled:

```
no handler for exception:
no networks where registered, please verify your ostore network
configuration: <err_0001_0141>
```

The client tried to find the server over the network. With an incorrect setting for OS_ROOTDIR, this happens even when the client and server are on the same machine.

Defaults

The following are the defaults for OS_ROOTDIR according to platform:

- AIX: /usr/lpp/ODI/ostore
- Solaris: /opt/ODI/ostore
- Other UNIX platforms: /usr/local/ODI/ostore
- Windows: The parent of the directory containing the O6LOW.DLL that is executing

Windows

If ObjectStore was not installed in C:\OSTORE, you must set OS_ROOTDIR to the location of the directory in which ObjectStore was installed. OS_ROOTDIR should be kept in the DOS environment so that makefiles and DOS utilities can reference it.

OS_SCHEMA_KEY_HIGH

OS_SCHEMA_KEY_LOW

Default: 0

The OS_SCHEMA_KEY_HIGH environment variable specifies the high 4 bytes of a 64-bit schema key.

The OS_SCHEMA_KEY_LOW variable specifies the low 4 bytes of a 64-bit schema key.

If you run certain ObjectStore tools and utilities on schema-protected databases, set OS_SCHEMA_KEY_LOW and OS_SCHEMA_KEY_HIGH to specify the schema keys of the databases to be accessed. The tools and utilities for which you must set these variables include

- oscompact
- osexschm
- ossevol
- ossg

- ossize
- osverifydb

Any ObjectStore application, including an ObjectStore tool or utility, can have a schema key that allows it to access a protected database with a matching key. Normally, you specify a key for an application programmatically. See objectstore::set_current_schema_key() in the *C++ API Reference*. OS_SCHEMA_KEY_HIGH and OS_SCHEMA_KEY_LOW are provided because it is not possible for you to set the schema key of a tool or utility programmatically.

You can, however, build an application that performs the same function as an ObjectStore utility by calling a member of the class os_dbutil, os_compact, or os_schema_evolution. This application can specify the schema key programmatically.

Deploying applications

If you are deploying an application, you need to know that some ObjectStore tools (such as ossg) cannot be invoked from the ObjectStore API. To allow your customers to use such a tool on a database that you have protected, build an application that spawns the tool as a child process. Specify the key of the child process by setting the environment variables from within the application.

OS_SCHEMA_KEY_HIGH and OS_SCHEMA_KEY_LOW determine an application's schema key when an ObjectStore application attempts to access data in a schema-protected database and either of the following is true:

- The application did not set the schema key using objectstore::set_current_schema_key().
- The application's most recent call to objectstore::set_current_schema_key() specified 0 for both arguments.

Remember that when these environment variables determine an application's schema key, all schema-protected databases that the application accesses must have the same schema key.

If you run an application on a schema-protected database and the application does not have a schema key, or the application's schema key does not match the database's schema key, ObjectStore signals err_schema_key and issues an error message such as the following:

```
Error using schema keys
<err-0025-0151> The schema is protected and the key, if provided, did
not match the one in the schema of database db1.
```

For information about the schema protection API, see objectstore::set_current_schema_key(), os_database::change_schema_key(), and os_database::freeze_schema_key() in the *C++ API Reference*.

OS_SCHEMA_PATH

Default: not set

The OS_SCHEMA_PATH variable specifies an alternative path by which an ObjectStore application or DLL locates its associated schema database. The path to the schema database is written into the executable or DLL when it is built. If ObjectStore cannot find the schema database at that location, it searches in a list of directories specified

by OS_SCHEMA_PATH. To specify more than one directory path, separate the paths with semicolons.

See also ossetasp on page 221 for information on how to permanently store a new path in an executable or DLL.

OS_SECURE_RPC_DOMAIN

Default: not set

The OS_SECURE_RPC_DOMAIN variable specifies a local domain name. You use the variable with AUTH_DES authentication. If there is no system getdomainname() routine or if it returns null, this variable is consulted for the name of the local domain.

OS_SERVER_DEBUG_FILE_MAX_LINES

Default: Not set

This environment variable affects the server.

The OS_SERVER_DEBUG_FILE_MAX_LINES variable specifies the size of the server output log file in lines written to the file. When the number of lines in the file reaches the limit, the current file is renamed with the extension .old and a new server output log file is created.

OS_SERVER_OUTPUT_LOG_NAME

Default: Not set

This environment variable affects the server.

The OS_SERVER_OUTPUT_LOG_NAME variable specifies the name of the server output log file to use instead of the default name. If this variable is not set, the log file is assigned the default name of oss_out on UNIX or osserver.txt on Windows, in the temporary directory.

OS_SNDBUF_SIZE

Default: 16384 bytes

The OS_SNDBUF_SIZE variable sets the default size of the network buffer used by the client to transmit data to the server. For best results, this size should be the same as the value of the server parameter Preferred Network Receive Buffer Size.

See Preferred Network Receive Buffer Size on page 95 for additional information about the server network buffer size. See further discussion about these parameters and the ways they affect performance in OS_RCVBUF_SIZE on page 121.

OS_STDOUT_FILE (Windows Only)

Default: Not set

The OS_STDOUT_FILE variable specifies the path of a file to which you want to redirect output that ObjectStore would otherwise send to stdout and stderr.

If your application is a Windows GUI and this variable is not set, ObjectStore displays the output in a message box.

To separate the output from several applications, you can set this variable to a path name such as `C:\TEMP\DEBUG.%d`. ObjectStore substitutes the process ID in place of the `%d`.

OS_TIX_BUFFER_SIZE (Windows Only)

Default: 8192 bytes

This environment variable affects the server, client, and cache manager.

The `OS_TIX_BUFFER_SIZE` variable specifies the size in bytes of the error report buffer. The default size is large enough for all reasonable and expected errors. However, if an error message is extremely long, it might overflow the buffer and cause the application to abort. If this happens, you can resolve the problem by setting this variable to a larger value.

OS_TIX_WD (UNIX Only)

Default: Not set

This environment variable affects the server, client, and cache manager.

The `OS_TIX_WD` variable specifies the working directory for ObjectStore when an unhandled TIX exception causes ObjectStore to create a core dump. (Whether there is a core dump depends on the platform and on the setting of the `OS_DEF_EXCEPT_ACTION` environment variable.)

When you specify a directory for `OS_TIX_WD`, ObjectStore sets that directory to be the working directory before it creates the core file.

This is particularly useful for ObjectStore daemons. For example, on UNIX systems the cache manager always runs with its working directory set to the `root` directory. This avoids administration problems such as preventing volumes from being unmounted. Set the `OS_TIX_WD` variable to control where ObjectStore puts core dumps of the cache manager daemon.

OS_TMPDIR

Default: Not set

The `OS_TMPDIR` variable specifies a directory in which to place temporary files. If you do not set `OS_TMPDIR`, ObjectStore uses the path returned by the Win32 function `GetTempPath()` on Windows or `/tmp` on UNIX. To set this variable, specify any nonnull value except `0`.

OS_TRACE_MISSING_VTBLS

Default: `false`

The `OS_TRACE_MISSING_VTBLS` variable causes a run-time debugging message to be printed to `stderr` when a missing vtbl handler is installed for a class. The message identifies the class with the missing vtbl handler. For example:

```
Installing missing vtbl: Class: CCC Name: NNN Symbol: SSS
```

Where

- `CCC` identifies the class of the top-level object for which the vtbl was found to be missing.
- `NNN` identifies the independent name of the vtbl; for example, `Derived_class::Base_class`.
- `SSS` identifies the mangled name of the vtbl symbol that was missing.

When you are missing a vtbl, the `err_missing_vtbl` run-time error message does not indicate the vtbl that is missing. If you rerun the application with `OS_TRACE_MISSING_VTBLS` turned on, ObjectStore catalogs vtbls that

- Were not found at initialization
- Might signal `err_missing_vtbl` errors at a later time

When you are debugging problems involving missing vtbls, turning on this variable helps determine those classes that need vtbls. Any nonempty value except `0` enables the debugging message.

OS_TURN_ON_ENGLISH_MESSAGES

Default: `false`

This environment variable affects the server, client, and cache manager.

When set to `1`, the `OS_TURN_ON_ENGLISH_MESSAGES` variable forces the printing of English messages along with the catalog-retrieved language-specific message.

OS_USE_MAP_FIXED

Default: Not set

This environment variable is used on Solaris and Linux platforms to specify that ObjectStore will always use the address range for the persistent storage region (PSR) specified by the `OS_AS_START` environment variable. On Linux platforms, `OS_USE_MAP_FIXED` is required if you use the `OS_AS_START` environment variable. On Solaris platforms, `OS_USE_MAP_FIXED` is optional. The Solaris operating system checks whether the address range specified for the PSR by the `OS_AS_START` environment variable is available, and if it is not, reports an error. If you know by means of other diagnostic tools that the specified range really is available, you can use `OS_USE_MAP_FIXED` to override the operating system check and place the PSR in the specified range.

For more information, see OS_AS_START on page 104.

OS_USER_ARCH_SET

Default: Not set.

The `OS_USER_ARCH_SET` variable allows users to define their own architecture sets. The name of an architecture set is specified as the argument to `ossg`'s `-arch` option, as described in ossg.

The `OS_USER_ARCH_SET` variable accepts strings of the form

`set_name1``(`*`arch_name1`* `[`*`arch_name2`* `...]) [`*`set_name2`* `...]`

where *`set_name`* is a user-defined name of an architecture set and *`arch_name`* is a predefined name of a platform that is a member of *`set_name`*. For a list of the predefined names of platforms, invoke `ossg` with the `-showsets` option, as described in ossg.

Caution

Do not use this environment variable to define names of architecture sets, unless directed by Technical Support. Only the standard architecture sets (`all`, `all32`, and `all64`) and versioned architecture sets are supported. For information about architecture sets, see ossg Neutralization Options on page 89 in Chapter 5, Building Applications for Multiple Platforms, on page 81 of *Building C++ Interface Applications*.

OS_USERNAME

Default: Not set.

The `OS_USERNAME` variable specifies a username to be used by a client application to connect to an ObjectStore server; used with the `OS_PASSWORD` environment variable. If the client allows `Name Password` authentication, the client connects to the server without prompting for a username and password.

OS_VERIFYDB_BUFFER_SIZE

Default: 512 MB

The OS_VERIFYDB_BUFFER_SIZE variable specifies the maximum size of the transient buffer used to optimize locality of reference by osverifydb in fast mode.

OS_VERIFYDB_FAST

Default: Not set.

The OS_VERIFYDB_FAST variable specifies that you want the osverifydb utility to always run in fast mode. Fast mode uses techniques that are optimized for performance purposes. Alternatively, you can run osverifydb with the -F option to specify fast mode.

If you set the OS_OSVERIFYDB_FAST environment variable to true (non-zero), osverifydb runs in fast mode by default.

You can use the OS_OSVERIFYDB_BUFFER_SIZE environment variable to specify the maximum size of the transient buffer used to optimize locality of reference by osverifydb in fast mode. The default size is 512 MB.

When you run the osverifydb utility, the default ObjectStore client cache size is 256 MB. The normal default cache size is 8 MB. You can override the default cache size by setting the OS_CACHE_SIZE environment variable.

The fast mode for osverifydb is not compatible with the following options:

- -ignore_references
- -illegal_pointer_action
- -o
- -v
- -whohas

Chapter 4
Utilities

This chapter provides information about ObjectStore utilities. Many of these utilities are implemented using the `os_dbutil` class functions. See the *C++ API Reference* for more information about the functions.

Note that the path of the executable for an ObjectStore utility is

- UNIX: `$OS_ROOTDIR/bin/`*`utility_name`*
- Windows: `%OS_ROOTDIR%\bin\`*`utility_name`*`.exe`

The following utilities are described in alphabetical order in this chapter:

The following table alphabetizes the different tasks you might need to perform when managing ObjectStore. The second column lists the ObjectStore utilities to use when performing each task.

Task	*ObjectStore Utility*
Backing up a database	osbackup on page 143
Changing database affiliations	osaffiliate on page 135
Changing database group names	oschgrp on page 150
Changing rawfs link hosts	oschhost on page 151
Changing database permissions	oschmod on page 152
Changing database owners	oschown on page 154
Compacting databases	oscompact on page 160
Comparing schemas	osscheq on page 215

Task	*ObjectStore Utility*
Configuring ObjectStore	`osconfig` on page 164
Copying databases	`oscopy` on page 167
Creating a rawfs directory	`osmkdir` on page 195
Creating a rawfs link	`osln` on page 190
Disconnecting a client thread on a server	`ossvrclntkill` on page 242
Displaying cache manager status	`oscmstat` on page 157
Displaying class names in a schema	`osexschm` on page 180
Displaying database host name	`oshostof` on page 185
Displaying used and available rawfs disk space	`osdf` on page 172
Displaying version information for ObjectStore	`osversion` on page 263
Dumping C++ databases	`osdump` on page 173
Dumping Java databases	`osjidump` on page 186
Evolving schemas	`ossevol` on page 224
Expanding file names	`osglob` on page 184
Freeing unreachable storage	`osgc` on page 181
Generating schemas	`ossg` on page 229
Getting server and client statistics	`ossvrstat` on page 248
Listing a directory	`osls` on page 194
Loading a C++ database from a dump	`osload` on page 192
Loading a Java database from a dump	`osjiload` on page 188
Logging transactions between backups	`osarchiv` on page 137
Measuring Database Size	`ossize` on page 238
Moving databases offline and online	`osdbcontrol` on page 169
Moving a directory, database, or link	`osmv` on page 196
Patching an executable with application schema path names	`ossetasp` on page 221
Pinging a server	`ossvrping` on page 245
Propagating data from the transaction log to the database	`ossvrchkpt` on page 241
Propagating data from the transaction log to the database (ObjectStore / Single)	`osprop` on page 198
Removing cache and commseg files	`oscmrf` on page 155
Removing a database or rawfs link	`osrm` on page 213
Removing a directory	`osrmdir` on page 214
Restoring a database from an archive log	`osrecovr` on page 199
Restoring a database from backup	`osrestore` on page 208

Task	*ObjectStore Utility*
Replicating a database	`osreplic` on page 205
Setting a remote schema path name	ossetrsp
Setting the server debug trace level	`ossvrdebug` on page 244
Shutting down the cache manager	`oscmshtd` on page 156
Shutting down the server	`ossvrshtd` on page 246
Starting the Server	`osserver` on page 217
Testing a Path Name for Specified Conditions	`ostest` on page 257
Verifying Pointers and References in a Database	`osverifydb` on page 258

osaffiliate

The `osaffiliate` utility displays or changes the external database affiliations for the specified database. When used to change the affiliations, this utility changes the path to the affiliated database; it does not change data or pointers created by the user.

Syntax

```
osaffiliate [options] db_path [index] [target_db_path]
```

Arguments

`db_path`

The path of the database whose path to an external database you want to view or change.

`index`

The index number associated with an external database as found in the affiliation table of the database specified by *`db_path`*.

`target_db_path`

The canonical path of the external database that you want to affiliate with the database specified by *`db_path`*.

Options

`-absolute`

Causes the complete path of the target database (including server prefix) to be stored in the affiliation table. This option overrides the default of storing a relative path from the source database to the target database. This option cannot be combined with the `-parent` option.

If osaffiliate is run with the -absolute option and the target pathname contains the fully qualified domain name of the target host, the affiliation will also have the fully qualified domain name.

`-force`

If *`target_db_path`* does not exist, `-force` causes a temporary version of *`target_db_path`* to be created to update the affiliation table. After the affiliation table is updated, the temporary version of *`target_db_path`* is destroyed. If *`target_db_path`* exists, the target database is left unchanged.

`-parent` *`parent_dir`*

Forces the relative path to traverse through the *`parent_dir`* even if that directory is not the shortest path. The *`parent_dir`* must be a prefix of both the source and target database canonical paths. This option cannot be combined with the `-absolute` option.

Description

When an ObjectStore database refers to objects in another (external) ObjectStore database, the external database is *affiliated* with the database containing the cross-database references. The ObjectStore database keeps the location information for the external databases in an affiliation table.

If you change the location of some of the files relative to each other, you probably will need to use the `osaffiliate` utility to change the affiliation table entries so they point to the correct new locations.

The affiliation table lists the index values of the external databases affiliated with the database specified with *`db_path`*. Use the following syntax to display the affiliation table:

```
osaffiliate db_path
```

Then run `osaffiliate [options] db_path index target_db_path` for each affiliated database for which you need to specify a new location.

API

See `os_database::change_affiliation()` in the *C++ API Reference*.

Examples

```
osaffiliate d:\db\rentals.db
```

This example displays the affiliation table of `rentals.db` as follows:

```
Affiliations of "D:\db\rentals.db":
Index 0: <reserved>
Index 1: <reserved>
Index 2: "common\inventory.db"
```

In this case, an external database (`inventory.db`) has an index value of `2`.

The next example changes the affiliation of the external database referred to by `rentals.db` from its old location (`common\inventory.db`) to a new location indicated by `shared\inventory.db`.

```
osaffiliate d:\db\rentals.db 2 shared\inventory.db
```

osarchiv

The `osarchiv` utility records all transaction activity for specified databases. You can run this utility interactively or in the background. See also Overview of the Backup/Restore Facility on page 48.

Syntax

```
osarchiv [options] -d directory [pathname ...]
```

Argument

`pathname ...`

Specifies a database, or a rawfs directory containing databases, whose transactions you want to log. You can specify one or more paths, and paths can be on different servers.

When you specify a rawfs directory, `osarchiv` logs transactions for all databases in that directory. It does not operate on databases that are in subdirectories unless you specify the `-r` option.

The group of databases for which you are performing archive logging is called the *archive set*.

If you do not specify at least one path name, you must specify the `-I` option with an import file name.

Options

`-a` *`archive_record_file`*

Optional. Specifies the path of the file that `osarchiv` uses to record the change IDs for the archive set. The `osarchiv` utility updates this file each time it successfully records committed changes to the archive set. (This process is referred to as *taking a snapshot*.) The archive record file is comparable to the incremental record file for `osbackup`.

`-B` *`size`*

Optional. Specifies the size of the buffer used by each server that `osarchiv` contacts. *`size`* is a number optionally appended with `k`, `m`, or `g` to indicate kilobytes, megabytes, or gigabytes, respectively. If no letter is specified, `m` is presumed. For example, `-B 1024k`, `-B 1m`, and `-B 1` each specify a maximum buffer size of 1 MB. The default value is 1 MB.

`-C`

Enables the interactive command-loop feature. This feature is disabled by default. See Commands on page 139 for the commands available in interactive mode.

`-c`

Optional. Provides a clean-up handler if you use Control-C to exit the utility.

`-d` *`directory`*

Required. Specifies the directory in which to create the archive log files.

`-i` *`interval`*

Optional. Specifies an integer that `osarchiv` uses as the interval between snapshots. By default, this interval is in seconds, but you can append `m`, `h`, or `d` to the value of the interval to indicate minutes, hours, or days. For example, `-i 60` and `-i 1m` both specify an interval of one minute.

When *`interval`* is not `0`, `osarchiv` takes a snapshot immediately after being initiated, then every *`interval`* seconds (or minutes, hours, or days) thereafter.

When you do not specify an interval, it defaults to `0`, which means that snapshots are not taken automatically. You can take a snapshot at any time that `osarchiv` is active by issuing the `x` command. See the command description for `x`.

`-I` *`import_file`*

Optional. Specifies the name of a file that contains a list of either file or rawfs database path names. The `osarchiv` utility logs transactions for the databases in this list. The `osarchiv` utility cannot read such a list from `stdin`.

The list contains one path name per line. Leading and trailing white space are ignored.

If you specify the `-I` option, you can also specify additional path names on the command line. After you initiate the `osarchiv` utility, you can use the `a` command to add databases to the archive set. See `a pathname`. You cannot specify `-I-`.

`-o` *`output_file`*

Optional. Redirects output to *`output_file`*.

`-P`

Optional. Specifies that `osarchiv` will use data compression.

`-r`

Optional. Instructs `osarchiv` to descend into subdirectories of any rawfs directories specified on the command line, adding all rawfs databases found to the archive set. By default, only databases in the specified directory are backed up.

When you are archiving file databases, specifying the `-r` option has no effect. You must specify each file database explicitly.

After archive logging begins, you can add a rawfs directory to the archive set. If you specified `-r` when you initiated `osarchiv`, it applies to subsequently added rawfs directories.

You cannot specify the `-r` option for some directories and not for others. When `-r` is specified, it applies to the entire archive set.

`-s` *`size`*

Optional. Specifies the maximum amount of data to write to an archive file. By default, this size is in megabytes. You can specify KB, MB, or GB by appending `k`, `m`, or `g` to *`size`*. For example, `-s 1024k`, `-s 1m`, and `-s 1` each specify a maximum archive file size of 1 MB.

When an archive file is full, the `osarchiv` utility automatically starts using the next file in the archive file sequence. A particular snapshot is always in a single archive file; `osarchiv` never stores it across two files.

The default is 2 MB.

`-T` *`seconds`*

Optional. Specifies the network timeout value. The value *`seconds`* is the amount of time. By default the value is the number of seconds, but `m`, `h`, or `d` can be appended to to the value to specify minutes, hours, or days, respectively. For example, `-T 60` and `-T 1m` both specify a timeout value of one minute. The default timeout value is 10 hours.

Commands

You can execute the following commands when you use `osarchiv` in interactive mode. The utility processes the commands between snapshots.

`a` *`pathname`*

Adds the specified file database or rawfs database or directory to the archive set.

`h`

Displays online help.

`i` *`interval`*

Interval — changes the interval between snapshots. Specify an integer for *`interval`*. You can append the letter `m`, `h`, or `d` to indicate minutes, hours, or days. For example, `i 60` and `i 1m` both specify an interval of one minute. When *`interval`* is `0`, snapshots are not taken automatically.

You can specify `i` without an integer to display the current interval.

`n`

Next — closes the current archive file and starts saving snapshots in the next archive file in the sequence.

`q` or `EOF`

Quit — takes a snapshot immediately and then terminates the `osarchiv` utility.

`r` *`pathname`*

Removes the specified file database or rawfs database or directory from the archive set.

`t`

Table of contents — displays the path names of the databases and rawfs directories in the archive set.

`x`

eXplicit — takes a snapshot as soon as you issue the command. This command has no effect on snapshot intervals.

Description

When archive logging is active, ObjectStore takes snapshots of modifications to the archive set. An archive snapshot records all data modified by transactions that have committed since the last snapshot was taken.

When a transaction commits, its modifications are not saved in the archive until the next snapshot.

When you start `osarchiv`, the first snapshot records data modified by transactions that committed since the last time the `osbackup` or `osarchiv` utility was run.

If a failover occurs while `osarchiv` is running, the utility will restart using the last snapshot and continue to write to the current archive log file.

Tape device

You cannot perform archive logging to a tape device.

Archive file format

The `osarchiv` utility places snapshots in archive files in the directory that you specified when you initiated the `osarchiv` utility. The utility uses the following naming convention for archive files:

`YYYYMMDDHH.ext`

Variable	*Meaning*
`YYYY`	Year
`MM`	Month
`DD`	Day
`HH`	Hour
`ext`	Extension of the form `aaa`, `aab`, `aac`, and so on

Switching archive files

The `osarchiv` utility places consecutive snapshots in the same archive file until one of the following happens:

- You issue the `n` command, which instructs `osarchiv` to use the next archive file.
- An archive file contains the maximum amount of data allowed (specified with `-s`) and `osarchiv` switches to the next archive file in the sequence. The default maximum size is 2 MB.

Ensuring sufficient disk space

You must ensure that there is sufficient disk space available to the `osarchiv` utility by periodically moving archive files to secondary storage. When `osarchiv` runs out of disk space for archive files, it notifies you and suspends activity. You must move archive files or allocate additional disk space to allow the utility to continue.

Adding to archive set

When you add a database to a directory for which you are performing archive logging, the `osarchiv` utility does not begin to take snapshots of that database automatically. To enable archive logging for the additional database, you must use the `a` command to explicitly add the database to the archive set.

Deleting a database

When you are performing archive logging for a database, the server keeps the database open. This has implications for deleting databases.

Windows

On Windows, you cannot delete a database for which you are performing archive logging until you invoke the `osarchiv r` command to remove the database from the archive set.

UNIX

On UNIX systems, when you remove a file, the operating system removes its directory entry but does not actually delete the file or free associated disk space until there are no applications with the database open. Again, you must invoke the `osarchiv r` command to remove the database from the archive set.

On all systems, the `r` command does not take effect until the end of a snapshot.

Tradeoffs for Obtaining the Results You Need

Decreasing the time between snapshots decreases the number of transactions recorded in each snapshot. Shorter intervals between snapshots have the effect of keeping the archive more up to date and keeping the amount of data that needs to be archived smaller.

However, each snapshot causes information to be written to the archive file, even if no data modifications are being recorded. Taking snapshots too frequently can consume space in the archive file unnecessarily. Longer intervals can reduce the amount of data being logged in cases in which the same data is modified by multiple transactions. In such cases, only the most recent copy of the committed data needs to be logged.

Examples

In the following example, `./inc` is the path name of the file that `osarchiv` uses to record the cluster change IDs for the archive set. The `osarchiv` utility updates this file each time it takes a snapshot. The directory in which to create the archive log files is `/vancouver1/archives`. The `-i` option indicates that snapshots should be taken every 30 seconds. The `-r` option instructs `osarchiv` to descend into any rawfs directories specified on the command line, adding to the archive set all rawfs databases found. Finally, `vancouver::/` specifies a rawfs directory whose transactions you want to log.

```
% osarchiv -C -a ./inc -d /vancouver1/archives/ -i 30 -r vancouver::/
Writing backup volume #1 (/vancouver1/archives/1999011216.aaa)...

Display archive set members:

> t
vancouver::/foo.db
vancouver::/dbdir/bar.db
vancouver::/dbdir/foo.db
```

Take a snapshot now:

```
> x
Archiving 452 sectors in database vancouver::/dbdir/bar.db.
Archiving 452 sectors in database vancouver::/dbdir/foo.db.
Archiving 452 sectors in database vancouver::/foo.db.
```

Add to the archive set:

```
> a /vancouver1/dbdir/foo.db
```

Display archive set members:

```
> t
vancouver::/foo.db
vancouver::/dbdir/bar.db
vancouver::/dbdir/foo.db
vancouver:/vancouver1/dbdir/foo.db
```

If you press Enter while the `osarchiv` utility is taking a snapshot, the utility displays a message such as the following. If it is not taking a snapshot, the utility displays another prompt symbol.

```
>
Archiving 452 sectors in database vancouver:/vancouver1/dbdir/foo.db.
```

Save snapshots in next archive file:

```
> n
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
Writing backup volume #2 (/vancouver1/archives/1999011216.aab)...
```

Display the snapshot interval:

```
> i
Snapshot interval is 5 seconds.
```

Change the snapshot interval:

```
> i 1m
> i
Snapshot interval is 60 seconds.
```

Remove a member of the archive set, and display archive set members:

```
> r vancouver::/foo.db
> t
vancouver::/dbdir/bar.db
vancouver::/dbdir/foo.db
vancouver:/vancouver1/dbdir/foo.db
```

Take a snapshot now:

```
> x
Archiving 68 sectors in database vancouver::/foo.db.
```

Take a snapshot and terminate the `osarchiv` utility:

```
> q
Closing volume #2 (/vancouver1/archives/1999011216.aab).
%
```

osbackup

The `osbackup` utility copies specified databases to another online location or to tape. See also Overview of the Backup/Restore Facility on page 48.

Syntax

```
osbackup [options] -i incremental_record_file
-f backup_image_file [-f backup_image_file]... pathname ...
```

Arguments

`pathname ...`

Specifies a database or rawfs directory to be backed up. You can specify one or more paths. Paths can be on different servers.

Options

-a

Optional. Aborts the backup operation if the utility cannot open the backup device; for example, if a tape is write protected or is not loaded. This option raises an exception that indicates the problem.

The default is that if the backup utility fails to open the backup device, it displays a message and waits for you to correct the problem.

`-b` *`blocking_factor`*

Optional. Specifies a blocking factor to use for tape input and output. The blocking factor is in units of 512-byte blocks. This parameter is ignored for regular files. The default on UNIX is 126 blocks. The maximum blocking factor is 512 blocks.

`-B` *`size`*

Optional. Specifies the size of the buffer used by the servers contacted by `osbackup`. *`size`* is a number optionally appended with `k`, `m`, or `g` to indicate kilobytes, megabytes, or gigabytes, respectively. If no letter is given, `m` is presumed. For example, `-B 1024k`, `-B 1m`, and `-B 1` each specify a maximum buffer size of 1 MB. The default value is 1 MB.

`-c`

Optional. Provides a clean-up handler if you use Control-C to exit the utility.

`-f` *`backup_image_file`*

Required. Specifies the location of the backup image.

You can specify a local file or a locally mounted file.

You can specify a tape device that is directly accessible from the host on which you are running `osbackup`. You cannot specify a remote tape device.

You can repeat the `-f` option with a new *`backup_image_file`* to create a multifile backup. When you do this, specify the `-s` option to indicate the size of each *`backup_image_file`*.

For example:

```
osbackup -s 1m -f back1 -f back2 -f back3 db1 db2 db3
```

ObjectStore tries to back up the databases to the `back1`, `back2`, and `back3` files. The utility prompts for additional file names if 1 MB per file is not sufficient.

On UNIX systems, you can specify `-f -` (hyphen) to indicate `stdout`. This allows you to pipe `osbackup` output directly to the `osrestore` utility.

On Windows systems, you specify a tape device with this syntax, `\\.\Tape0`, the standard Windows name for the first tape drive.

`-i` *`incremental_record_file`*

Required. Specifies the incremental record file, a file that contains information about those databases that have been backed up and when they were backed up. The `osbackup` utility uses this information to determine the clusters within a database that have been modified since the last backup at a lower level. The utility then backs up only modified clusters. The incremental record file is comparable to the archive record file for `osarchiv`.

Performing a backup at any level for which no previous information exists is equivalent to doing a level 0 backup for that database.

`-I` *`import_file`*

Optional. Specifies the name of a file that contains a list of either file or rawfs database path names. The `osbackup` utility backs up the databases in this list. If you specify `"-"` as the import file name, `osbackup` reads from standard input.

The list contains one path name per line. Leading and trailing white space is ignored.

If you specify the `-I` option, you can also specify additional paths on the command line.

`-l level`

Optional. Specifies the level of the backup. Specify an integer from 0 to 9. Files that have been modified since the last backup at a lower level are copied to the backup image. For example, suppose that you did a level 2 backup on Monday, followed by a level 4 backup on Tuesday. A subsequent level 3 backup on Wednesday would contain all files modified or added since the level 2 (Monday) backup.

Backup is incremental at the cluster level, meaning that a cluster is only backed up if it has been modified since the last backup at a lower level. A level 0 backup (the default) backs up all clusters in all specified databases.

`-o output_file`

Optional. Redirects output to *`output_file`*.

`-P`

Optional. Specifies that `osbackup` will use data compression.

`-r`

Optional. Instructs `osbackup` to descend into subdirectories of any rawfs directories specified on the command line, adding all rawfs databases found to the list of databases to be backed up. By default, only databases in the specified directory are backed up. When backing up file databases, specifying the `-r` option has no effect. You must explicitly specify each file database.

`-s size`

Optional. Sets the size of the volume being dumped to. The `osbackup` utility prompts you to insert a new tape or specify a new backup image file after it writes the amount of data specified by *`size`*.

You can specify `k`, `m`, or `g` to indicate that *`size`* is in units of kilobytes, megabytes (the default), or gigabytes. For example, `-s 1024k`, `-s 1m`, and `-s 1` each specify a maximum backup image size of 1 MB.

You can use this option with the `-f` option to perform a multivolume backup.

This option is mainly for use when you are backing up to a tape device, because end-of-media cannot be detected reliably on some systems.

On Solaris 2, the `-s` option is not required because the end of the tape is reliably signaled to the application without any loss of data. On other systems, if you do not specify `-s`, the `osbackup` utility terminates when it reaches the end of the tape.

`-S exec_command_name`

Optional. Specifies the path of a command to be executed when the `osbackup` utility reaches the end of the media. This command should mount the next volume before returning. The exit status from this command must be 0 or the backup operation aborts. Note that this option is an uppercase `S`.

`-T seconds`

Optional. Specifies the network timeout value. The value *`seconds`* is the amount of time. By default the value is the number of seconds, but `m`, `h`, or `d` can be

appended to to the value to specify minutes, hours, or days, respectively. For example, `-T 60` and `-T 1m` both specify a timeout value of one minute. The default timeout value is 10 hours.

Description

When backing up databases, ObjectStore takes advantage of any operations already being performed by the server on behalf of various client applications. This reduces the cost of performing the backup. The `osbackup` utility gives priority to databases that are already open at the time the backup starts and, within a database, to those sectors that are being actively used.

When backup starts, `osbackup` determines those clusters that require backup, builds a map that describes this data, and sets itself up to intercept read and write requests to and from these sectors. Any time the server reads a sector of interest to the backup process that has not already been backed up, `osbackup` allows the read to proceed and makes a copy of the data at that time. Similarly, write requests are intercepted and delayed long enough for `osbackup` to retrieve the transaction-consistent data first. Otherwise, the backup process operates in the background, retrieving data as efficiently as possible.

If a failover occurs while `osbackup` is backing up a database, the utility restarts from the beginning.

Considerations

You can mix file databases and rawfs databases in the set of databases to be backed up.

- Backing up a rawfs directory

  When you specify a rawfs directory, `osbackup` backs up all databases in the directory. When you specify the `-r` option, `osbackup` also backs up all databases in all subdirectories, subsubdirectories, and so on.

- Backing up file databases

  When backing up file databases, you must explicitly specify the name of each database with the *`pathname`* argument or in an import file, specified with the `-I` option.

- Specifying an incremental record file

  If you do not specify an incremental record file for your backup, `osbackup` creates one with a default path name.

  If a file of this name already exists, it is written over and data in it is lost. For this reason, it is recommended that you use the `-i` option to provide a unique name for the incremental record file.

- Compacted databases

  When you run the `oscompact` utility on a database, it has the potential to modify each cluster in the database. When you back up a database after compacting it, the `osbackup` utility copies each modified cluster; this might be the entire database. Consequently, you might want to compact databases before you perform a full backup.

Examples

```
% osls -l vancouver::/foo.db
-rw-rw-r-- smith odi     231424 Dec 20 16:17 vancouver::/foo.db
%
```

Full backup of a rawfs database to three files:

```
% osbackup -i ./inc -f ./s1 -f ./s2 -f ./s3 -s 80k vancouver::/foo.db
Writing backup volume #1 (./s1)...
Archiving 452 sectors in database vancouver::/foo.db.
Closing volume #1 (./s1).
Auto switching to volume #2 (./s2).
Writing backup volume #2 (./s2)...
Closing volume #2 (./s2).
Auto switching to volume #3 (./s3).
Writing backup volume #3 (./s3)...
Closing volume #3 (./s3).
%
```

If you do not specify enough files, you are prompted as follows:

```
% osbackup -i ./inc -f ./s1 -f ./s2 -f ./s3 -s 10k vancouver::/mdltst1.db
Writing backup volume #1 (./s1)...
Closing volume #1 (./s1).
Auto switching to volume #2 (./s2).
```

```
Writing backup volume #2 (./s2)...
Closing volume #2 (./s2).
Auto switching to volume #3 (./s3).
Writing backup volume #3 (./s3)...
Archiving 913 sectors in database vancouver::/mdltst1.db.
Closing volume #3 (./s3).
Please enter the pathname of the next file to use for backup.
```

Full backup of a rawfs database to an existing image:

```
% osls vancouver::/
dbdir/
foo.db
% osls vancouver::/dbdir
bar.db
foo.db

% touch ./img <-- create file to demonstrate problem
% osbackup -f ./img -i ./inc vancouver::/foo.db

Error encountered while opening file ./img (File ./img already exists.
Cannot archive to an existing file.)

Do you wish to try_again? (yes/no): yes
Please enter the pathname of the next file to use for backup. ./img2
Writing backup volume #1 (./img2)...
Archiving 452 sectors in database vancouver::/foo.db.
Closing volume #1 (./img2).
```

Full backup of directory:

```
% osbackup -f ./img -i ./inc -r vancouver::/
Writing backup volume #1 (./img)...
Archiving 452 sectors in database vancouver::/dbdir/bar.db.
Archiving 452 sectors in database vancouver::/dbdir/foo.db.
Archiving 452 sectors in database vancouver::/foo.db.
Closing volume #1 (./img).
```

Full backup of the databases listed in the import file:

```
% cat ./import_file
vancouver::/foo.db
/vancouver1/dbdir/foo.db
%

% osbackup -f ./img -i ./inc -I ./import_file
Writing backup volume #1 (./img)...
Archiving 452 sectors in database vancouver:/vancouver1/dbdir/foo.db.
Archiving 452 sectors in database vancouver::/foo.db.
Closing volume #1 (./img).
%
```

Using an import file and specifying a path:

```
% osbackup -f ./img -i ./inc -I ./import_file vancouver::/dbdir/foo.db
Writing backup volume #1 (./img)...
Archiving 452 sectors in database vancouver:/vancouver1/dbdir/foo.db.
Archiving 452 sectors in database vancouver::/dbdir/foo.db.
Archiving 452 sectors in database vancouver::/foo.db.
Closing volume #1 (./img).
%
```

Incremental backups of a rawfs database:

```
% $OS_ROOTDIR/bin/osbackup -f ./img0 -i ./inc -l 0 vancouver::/foo.db
```

```
                Writing backup volume #1 (./img0)...
                Archiving 452 sectors in database vancouver::/foo.db.
                Closing volume #1 (./img0).
% $OS_ROOTDIR/bin/osbackup -f ./img1 -i ./inc -l 1 vancouver::/foo.db
                Writing backup volume #1 (./img1)...
                Closing volume #1 (./img1).
% $OS_ROOTDIR/bin/osbackup -f ./img2 -i ./inc -l 2 vancouver::/foo.db
                Writing backup volume #1 (./img2)...
                Closing volume #1 (./img2).
```

```
% osrm vancouver::/foo.db
```

Restoring from incremental backups:

```
% $OS_ROOTDIR/bin/osrestore -f ./img0
Recovering from volume #1 (./img0)...
Restoring 452 sectors to database "vancouver::/foo.db"
Recovered to time Tue Jan 12 15:50:10 1999

Do you wish to restore from any additional incremental backups?
(yes/no):
yes
Closing volume #1 (./img0).
Please enter the pathname of the next file from which to restore.
./img1
Recovering from volume #2 (./img1)...
Recovered to time Tue Jan 12 15:50:21 1999

Do you wish to restore from any additional incremental backups?
(yes/no):
yes
Closing volume #2 (./img1).
Please enter the pathname of the next file from which to restore.
./img2
Recovering from volume #3 (./img2)...
Recovered to time Tue Jan 12 15:50:41 1999

Do you wish to restore from any additional incremental backups?
(yes/no):
no
Closing volume #3 (./img2).
%
```

oschgrp

The `oschgrp` utility changes the group name of the specified databases and directories.

Syntax

```
oschgrp [-R][-f] group pathname ...
```

Arguments

group

Specifies a group name or group number in a group ID file.

pathname ...

Specifies the databases or directories whose group name you are changing. You can specify either rawfs paths of any kind or file database paths.

Options

`-f`

Forces execution. Errors are not reported.

`-R`

Specifies that ObjectStore should change the group name recursively for all specified directories. That is, it changes the group name for subdirectories and their contents, subsubdirectories and their contents, and so on.

Description

The `oschgrp` utility operates on rawfs databases and file databases.

When you specify a file database, you cannot specify a remote file-server host name in the path name of the file database. The `oschgrp` utility passes the operation to a local native utility. If you specify a remote file-server host name, ObjectStore informs you that you specified an illegal path name.

Wildcards

The `oschgrp` utility can perform wildcard processing using regular expression wildcards `*`, `?`, `{}`, and `[]`.

UNIX

On UNIX systems, when you are operating on a rawfs database, you must enclose the wildcard in quotation marks (" ") or precede it with a backslash (\) to keep the shell from interpreting the wildcard as a shell wildcard.

The `oschgrp` utility accepts a combination of rawfs path names and file path names.

The group ID file is `/etc/group`. You must be the owner of the database or be the superuser to use this utility.

API

See `os_dbutil::chgrp()` in the *C++ API Reference*.

oschhost

The `oschhost` utility changes the host that a link in the rawfs points to.

Syntax

There are two forms of the `oschhost` utility. Note that the second form does not include options.

```
oschhost [-f][-R] newhost pathname ...
oschhost [server_host] old_link_host new_link_host
```

Arguments

`newhost`

Specifies the name of the new host for the specified rawfs link.

`pathname ...`

Specifies one or more rawfs links.

`server_host`

Specifies the server on which you are running `oschhost`. When you do not specify this argument, ObjectStore runs the utility on the local host.

`old_link_host`

Specifies the name of the host the link currently points to.

`new_link_host`

Specifies the name of the new host the link will point to.

Options

`-f`

Forces execution. Errors are not reported.

`-R`

Specifies that ObjectStore should change the host recursively for all specified directories.

Description

The `oschhost` utility operates only on rawfs links. The `oschhost` utility changes only the host component of the rawfs symbolic link, or all links in the rawfs. The utility does not physically move any databases or directories.

You can use `oschhost` to update the rawfs after you restore an entire file system from one server to another. Use the first form of the utility to change specified links, that is, links with particular path names.

Use the second form of the utility to change all links on a particular host (*`server_host`*) that point to a specified host (*`old_link_host`*) so that they point to a new host (*`new_link_host`*).

UNIX

On UNIX systems, you must be the superuser to change the host for a rawfs.

API

See `os_dbutil::rehost_link()` and `os_dbutil::rehost_all_links()` in the *C++ API Reference*.

oschmod

The `oschmod` utility changes the permission mode for the specified databases and directories.

Syntax

```
oschmod [-R][-f] new_mode pathname ...
```

Arguments

new_mode

Specifies the new permission mode for the specified databases and directories.

pathname ...

Specifies the databases and directories whose permission you want to change. You can specify both rawfs and file path names.

Options

`-f`

Forces execution. Errors are not reported.

`-R`

Specifies that ObjectStore should change the permission recursively for all specified directories.

Description

To change the permission mode for a database, you must be the owner of the database or, on UNIX, the superuser.

The *new_mode* argument can be absolute or symbolic.

Absolute mode

An absolute mode is an octal number constructed from the OR operation performed on the following modes (note that *execute* is meaningful only for directories):

Mode	*Meaning*
400	Read by user.
200	Write by user.
100	Execute (search in directory) by user.
040	Read by group.
020	Write by group.
010	Execute (search) by group.
004	Read by others.
002	Write by others.
001	Execute (search) by others.

Symbolic mode

A symbolic mode has the form

```
[who] op permission [op permission] ...
```

where *who* is a combination of the following:

Value	*Meaning*
u	User permissions
g	Group permissions
o	Others
a	All, or ugo

If you omit *who*, the default is a, but the setting of the file creation mask (on UNIX, see umask in sh(1) or csh(1) for more information) is taken into account. When *who* is omitted, oschmod does not override the restrictions of your user mask.

op is one of the following:

Value	*Meaning*
+	Add the permission.
–	Remove the permission.
=	Assign the permission explicitly (all other bits for that category, owner, group, or others, are reset).

permission is any combination of the following:

Value	*Meaning*
r	Read
w	Write
x	Execute

Omitting *permission* is useful only with =, to remove all permissions.

When you specify a file database, you cannot specify a remote file-server host in the path name of the file database. The oschmod utility passes the operation to a local native utility. If you specify a remote file-server host name, ObjectStore informs you that you specified an illegal path name.

Wildcards

The oschmod utility can perform wildcard processing using regular expression wildcards *, ?, {}, and [].

UNIX

On UNIX systems, when you are operating on a rawfs database, you must enclose the wildcard in quotation marks (" ") or precede it with a backslash (\) to keep the shell from interpreting the wildcard as a shell wildcard.

API

See os_dbutil::chmod() on page 177 in the *C++ API Reference*.

oschown

The `oschown` utility changes the ownership of specified databases and directories.

Syntax

```
oschown [-R][-f] owner[.group] pathname ...
```

Arguments

owner

Specifies the user name of the new owner of the specified databases and directories.

.group

Specifies the group name of the specified databases and directories. Be sure to precede it with a period. Optional.

pathname

Specifies the databases and directories whose owner you want to change. You can specify both file and rawfs path names.

Options

`-f`

Forces execution. Errors are not reported.

`-R`

Specifies that ObjectStore should change the owner recursively for all specified directories.

Description

This utility operates on rawfs databases and directories and on file databases and directories.

When you specify a file database, you cannot specify a remote file-server host in the path name of the file database. The `oschown` utility passes the operation to a local native utility. If you specify a remote file-server host name, ObjectStore informs you that you specified an illegal path name.

Wildcards

The `oschown` utility can perform wildcard processing using regular expression wildcards `*`, `?`, `{}`, and `[]`.

UNIX

On UNIX systems, you must be the superuser to run this utility. The owner must be a user name in the password file `/etc/passwd`. Only the superuser can change the owner of a directory or database. The group is a group name found in the GID file `/etc/group`.

When you are operating on a rawfs database, you must enclose the wildcard with quotation marks (" ") or precede it with a backslash (\) to keep the shell from interpreting the wildcard as a shell wildcard. The `-f` and `-R` options are identical to the shell `chown` command's force and recursive options, respectively. The `oschown` utility accepts a combination of rawfs path names and file path names.

API

See `os_dbutil::chown()` on page 177 in the *C++ API Reference*.

oscmrf

The `oscmrf` utility instructs the cache manager on the specified host to delete the cache files and commseg files in its free pool.

Syntax

```
oscmrf [hostname]
```

Argument *hostname*

Specifies the host of the cache manager that you want to instruct to delete cache and commseg files. The default is the local host.

Description

It is always safe to run the `oscmrf` utility. The cache manager deletes only files that are not in use by any client.

After `oscmrf` runs, if an additional client appears, the cache manager must create new cache and commseg files. This is slightly slower than if it did not have to create these files.

Windows

The cache manager does not use cache files or commseg files on Windows systems. However, you can use the `oscmrf` utility on these operating systems and specify hosts that do use cache and commseg files.

API

See `os_dbutil::cmgr_remove_file()` on page 178 in the *C++ API Reference*.

Example

```
% oscmrf
Deleted 2 cache files and 2 commseg files.
%
```

oscmshtd

The `oscmshtd` utility shuts down the cache manager on the specified host.

Syntax

```
oscmshtd [hostname] [version]
```

Arguments

`hostname`

Specifies the host of the cache manager that you want to shut down. The default is the local host.

`version`

Specifies the version of the cache manager that you want to shut down. The default is `6`.

Description

Be sure to notify users before you shut down the cache manager.

API

See `os_dbutil::cmgr_shutdown()` on page 178 in the *C++ API Reference*.

Example

```
%oscmshtd
Shutting down Cache Manager process
%
```

oscmstat

The `oscmstat` utility displays status information about the cache manager process running on the specified host.

Syntax

```
oscmstat [hostname] [version_number]
```

Arguments

`hostname`

Specifies the name of the host of the cache manager for which you want information. The default is the local host.

`version_number`

Specifies the version of the cache manager for which you want information. The default is `6`.

Description

The information provided by the `oscmstat` utility is useful for debugging the storage system.

The `oscmstat` utility prints information about every server to which the cache manager is connected. For each server, it displays

- The name of the server host
- The client process ID of the client being processed, or `0` if none is being processed
- Information on notifications queued for clients

UNIX

If the cache manager is running on a UNIX system, `oscmstat` also displays the names of cache and commseg files managed by the cache manager. This is useful in determining whether files are in active use by ObjectStore or are free and available to be deleted using the `oscmrf` utility.

The information for the cache and commseg files includes the size of the files, the directory used to store them, the version of ObjectStore, and the name of the host that created and owns or owned the file. In the example shown below, the host name is `kellen`.

You can safely delete files labeled “Free” using `oscmrf`.

If `oscmstat` reports that no cache manager is running, it is safe to delete the files as long as you are certain that `oscmstat` did not fail due to temporary network failure or something similar.

API

See `os_dbutil::cmgr_stat()` on page 178 in the *C++ API Reference*.

Examples

Output for a cache manager on a UNIX platform typically looks like the following. (A cache manager on Windows systems uses no cache or commseg files, so there is no reference to them in the `oscmstat` output.)

```
kellen% oscmstat

ObjectStore Release 6.0
Cache Manager:
  Host: kellen
  PID:      11679 [1]
  Executable:      /os/top/lib/oscmgr6 [2]
  Version:      6.0
  Started at:      Fri Jan 22 14:35:40 1999
  Soft Allocation Limit:      none [3]
  Hard Allocation Limit;      none
  Used Temp File Space:      8.04MB [4]
  Free Temp File Space:      1.91MB
Free Cache Files: [5]
  1.91MB  /tmp/wirth/ostore/objectstore_6_kellen_cache_1
In-Use Cache Files:
  8MB  /tmp/ostore/objectstore_6_kellen_cache_109
Free Commseg Files:
  8KB  /tmp/wirth/ostore/objectstore_6_kellen_commseg_2
In-Use Commseg Files:
  40KB  /tmp/ostore/objectstore_6_kellen_commseg_110
Client Connections: [6]
  PID  UID  Commseg  Name & Version
  272  30263  0x00000000  /h/mangle/3/mj/tests
    /mj.houdini.sol2c4.picsym.2/obj/tscoll/houdini/colltest
  -dbname colltest.db -lgcard 100 (6.0)
Client Notifications: [7]
  Client  Queue  Received  Sent to  Pending  Overflows
  PID  Size  by cmgr   Client
  272  51  0  0  0  0
Server Connections: none
kellen00
kellen%
```

Notes

[1] Operating system process ID of the cache manager process.

[2] The executable cache manager file that is accessed by `oscmstat`.

[3] The allocation limit parameters are as described in the cache manager parameter file. See Setting Cache Manager Parameters on page 307. `oscmstat` does not display this information for cache managers running on Windows systems.

[4] Total sizes of the Free and In-Use cache and commseg files. `oscmstat` does not display this information for cache managers running on Windows systems.

[5] Free and In-Use cache and commseg files known to the cache manager. Files labeled "Free" can be safely deleted with the `oscmrf` utility. `oscmstat` does not display this information for cache managers running on Windows systems.

[6] Clients (ObjectStore application processes) currently running on this host. For each client, `oscmstat` gives the operating system process ID, the user ID associated with the client process, the commseg address for the process (useful in debugging the cache manager), the name of the client (assuming that the client has called `objectstore::set_client_name()`), and the ObjectStore version number.

[7] Notification information for each client. The `oscmstat` output displays the following:

- Process ID (PID) of the client.
- Size of the queue.
- `Received by cmgr`, which is the number of notifications that were received from any servers and were addressed to this client process. Each such notification either went into the client's queue or was discarded because the queue was full (that is, it was an overflow notification). Both kinds are counted here. This number only increases.
- `Sent to Client`, which is the number of notifications sent to the client. This means that the client called `os_notification::receive()` and obtained the notification. This number does not include the overflow notifications or the notifications that are still in the queue. This number only increases.
- `Pending`, which is the number of notifications that are in the queue. This number can increase and decrease.
- `Overflows`, which is the number of notifications that were discarded during the lifetime of the client process because the queue was full. This number only increases.

oscompact

The `oscompact` utility removes deleted space in specified databases.

Syntax

```
oscompact -dbs_to_compact pathname ... [options]
```

Options

Numeric values can be in decimal or hexadecimal format. Hexadecimal values must be prefixed by the characters 0x.

`-dbs_to_compact` *`pathname ...`*

Specifies one or more databases to compact. If using cross-database pointers or ObjectStore references then all affiliated databases should be compacted in the same command.

`-address_space_release_interval` *`interval`*

Controls the frequency of address-space releases, in units of bytes. Specify a smaller value for more frequent releases, a larger value for less frequent. The default value is displayed when you run `oscompact with no options.`

This option is useful when you want to prevent compaction from running out of address space while executing transformer functions.

`-explanation_level` *`n`*

Specifies the level of debugging information output by `oscompact`. The argument *`n`* must be an integer between `0` and `4`, which have the following meanings:

- `0` — Do not output debugging information (default).
- `1` — Output phase-level information for each cluster, segment, and database that is compacted.
- `2` — Output information about all types in the database.
- `3` — Output information about objects with transformers.
- `4` — Output all possible debugging information.

Note that the levels of information are accumulative: each level includes all information at the previous levels.

`-malloc_size` *`size`*

Specifies the malloc size in KB for the relocation map. The default is 1024. This option is rarely used.

`-max_bytes_wasted_to_avoid_page_boundary` *`size`*

Specifies that you want compaction to avoid placing objects across page boundaries in the compacted database. This applies to objects that the utility moves to compact the database into a smaller amount of space. When compaction avoids a page boundary, the utility inserts a free region and leaves some bytes on the page unused.

Replace *`size`* with the maximum number of bytes that it is okay to leave unused on a page. When the utility would leave more than *`size`* bytes free in order to

avoid allocating an object across a page boundary, it does not avoid the page boundary. You can specify a value of `0` through `4095` for size. Avalue of `0` disables this feature. The default is that compaction does not avoid placing objects across page boundaries.

Note: In the context of this option, all pages are server pages, which are always 4K in size. It does not matter what platform the server is running on.

`-maximum_cluster_size` *`max_size_bytes`*

Specifies that clusters larger than that the size represented by *`max_size_bytes`* will be split into multiple clusters. The value of *`max_size_bytes`* is rounded up to the next multiple of 64 KB. The default is to split any cluster that exceeds 1600 MB.

Note that some ObjectStore collections manage an internal cluster which will not be split because the internal clusters maintain data structures that cannot be split across multiple clusters. Examples of collections that maintain internal clusters are `os_set`, `os_Set`, `os_bag`, `os_Bag`, `os_Dictionary`, and indexes.

`-memory` *`size`*

Specifies the maximum memory size in MB to use for the pointer relocation map. The default is half the main memory or half the virtual memory, whichever is smaller.

`-prefetch` *`prefetch_KB`*

Specifies the maximum size in KB of persistent pages to prefetch at one time. The default is 1024 KB which should provide good performance on most hardware.

`-secondary_psr_size` *`secondary_psr_size`*

Specifies the size in MB of virtual address space for secondary sessions. The default is 16 MB.

`-threads` *`n_threads`*

Specifies the number of threads to use. This option is available only if the `-work_db` option is also used. If the `-work_db` option is used, the default number of threads is 1.5 times the number of CPUs in the system (rounded up). If the `-work_db` option is not used, one thread is used.

`-work_db` *`pathname`*

This option is used when compacting a large set of databases that require a lengthy compaction. The option should be a path to a database that `oscompact` will use to store information so that an interruption of compaction can be resumed from the most recent consistent state. Using this option allows the compaction to be done in multiple smaller transactions rather than one large transaction, which can prevent the `osserver` transaction log from becoming too large and prevent `address_space_full` exceptions.

When you do not specify this option, ObjectStore performs the entire compaction operation in one transaction. When you specify the `-work_db` option, ObjectStore uses a separate transaction for each cluster. Therefore, you can use separate threads to concurrently work on separate clusters.

Description

ObjectStore databases consist of clusters containing persistent data. As persistent objects are allocated and deallocated in a cluster, internal fragmentation in the cluster can increase because of the presence of holes produced by deallocation. Of course, the ObjectStore allocation algorithms recycle deleted storage when objects are allocated; however, there might be a need to compact persistent data by squeezing out the deleted space. Such compaction frees persistent storage space so it can be used by other clusters. The oscompact utility removes deleted space in specified databases by compacting all C++ persistent data, including ObjectStore collections, indexes, and bound queries, and correctly relocates pointers and all forms of ObjectStore references to compacted data. After you compact a database, access is usually faster so performance improves. Once a database is compacted it cannot be de-compacted.

Note

The fast schema evolution internals are used for speedy compaction so seeing an `err_schema_evolution` exception coming from `oscompact` is normal.

You can use the `oscompact` utility on both file databases and rawfs databases.

Compacting file databases

The clusters in file databases are made up of extents, all of which are allocated in the space provided by the host operating system for the single host file. When there are no free extents left in the host file and growth of an ObjectStore cluster is required, the ObjectStore server extends the host file to provide the additional space. The compactor permits holes contained in clusters to be compacted for return to the allocation pool for the host file. This frees that space for use by other clusters in the same database. However, because operating systems provide no mechanism to free disk space allocated to regions internal to the host file, any such free space remains inaccessible to other databases stored in other host files. In other words, compacting a file database does not reclaim space for use by other databases.

To decrease the size of the file database after compaction, use the `oscopy` utility to copy the database. The newly copied database will not contain the free space reclaimed by the compactor.

Compacting rawfs databases

The ObjectStore rawfs stores all databases in a storage pool, on either one or more host files or raw partitions. Any space in a rawfs that is freed by the compaction operation can be reused by any cluster in any segment in any database stored in the rawfs.

Database locking

During compaction the database is locked and only applications running MVCC transactions can access the database. However, when running with the `-work_db` option the compaction is done in multiple transactions which can result in an MVCC application seeing an inconsistent view of data. For example, some objects might have been moved but if an ObjectStore collection containing pointers to those objects has not been transformed, the result is an inconsistent view.

Backing up compacted databases

Before compacting a database the database should be backed up to safeguard against unexpected results. After compacting a database the next backup for that database should be a full backup. The reason for this is that most clusters will be modified during compaction and running an incremental backup after that will not be practical.

Transformers

Some data structures become invalid as a result of compaction. The classic example of a data structure that might require user transformation is a hash table that hashes on the offset of an object within a cluster. Because compaction modifies these offsets, there is no way that such an implicit dependence on the cluster offset can be accounted for by compaction. Therefore, the compacted hash table becomes invalid. ObjectStore collections and indexes are transformed by compaction. Invocation of a user data transforms can only be done when using the C++ `os_compact()` API.

C++ interface API

See `os_compact::compact()` in the *C++ API Reference*.

Very large databases

See *Advanced C++ API User Guide*, Evolving or Compacting Very Large Databases.

Restrictions

You must observe certain data restrictions when you use the compactor:

C++ interface

Because the ObjectStore `retain_persistent_addresses` facility requires that persistent object locations within a cluster do not vary, any client application using this facility on a database being compacted should release and retain the addresses after compaction completes.

Schema key protection

If you are developing an application and will be running `oscompact` on a database protected with a schema key, ensure that the correct key is specified for the environment variables `OS_SCHEMA_KEY_LOW` and `OS_SCHEMA_KEY_HIGH`. If the correct key is not specified for these variables, `oscompact` fails. and ObjectStore signals

```
err_schema_key _CT_invalid_schema_key,
"<err-0025-0151> The schema is protected and the key provided did not
match the one in the schema."
```

Java interface

Applications must not retain references to non-exported objects in a database while it is being processed by the `oscompact` utility.

For more information about the `compact()` function, see `os_compact::compact()` in the *C++ API Reference*. For more information about protecting databases with schema keys, see Restricting Access with Schema Keys in the *C++ API User Guide*.

osconfig

The `osconfig` utility configures ObjectStore clients and servers on UNIX platforms. Although it can be used to modify server parameters after you have installed ObjectStore, it is most commonly used when you are installing ObjectStore.

Note

The `osconfig` utility is only for use on UNIX platforms. It cannot be used on Windows platforms.

Syntax

The `osconfig` utility can be invoked using the following command lines:

`osconfig server`

Configures an ObjectStore server and client on this machine. The server will be configured to use file databases. After configuration, this command line will check that the client can address the local server. If you want to check that the client can address remote servers, you must use the `osconfig check -host` command line (see below).

`osconfig rawfs`

Configures an ObjectStore server and client on this machine. The server will be configured to use rawfs and file databases. This command line will also create and configure the rawfs. After configuration, `osconfig` will check that the client can address the local server. If you want to check that the client can address remote servers, you must use the `osconfig check -host` command line (see below).

`osconfig client [-host` *`list_of_names`*`]`

Configures an ObjectStore client on this machine. The `-host` option (if specified) must be followed by a space-separated list of names of servers that this client can address. If the servers run on a cluster, the list should consist of logical host names, not node names. After configuration, `osconfig` will check that the newly configured client can address the specified servers. If you do not specify the -host option, this command line will only check that the client can address the local server.

`osconfig failover [-server` *`server1 server2`*`]`

Configures an ObjectStore server to use ObjectStore-managed failover on this machine and another machine that shares a disk with this machine. The `osconfig` utility will prompt you for the other machine on which the failover server is running. If you are configuring failover to use two logical servers, you must specify the `-server` option, followed by the names of the two logical servers. For information about failover, see Cluster-Managed Failover on page 288.

`osconfig cluster -logical_host` *`list_of_names`*

Configures an ObjectStore server to use cluster-managed failover. You must specify the `-logical_host` option, which must be followed by a space-separated list of one or two logical host names. For more information, see Cluster-Managed Failover on page 288.

`osconfig check [-host list_of_names]`

Verifies that the ObjectStore client can address the local server on this machine. If you specify the `-host` option, osconfig will verify that the client can address the list of servers specified after the option. The names in the list must be space-separated and can include logical servers, real servers, or logical hosts on a cluster.

`osconfig { server | rawfs | failover | cluster } -upgrade`

Upgrades an existing ObjectStore server with an updated version of ObjectStore. For more information, see Performing ObjectStore Upgrades on page 165.

Description

The `osconfig` utility is typically used when installing ObjectStore to configure servers and clients that you plan to use. But it can also be used if you later decide to add another server or client, or if you want to verify that a client can address a remote server. If you plan to use the same host as both a server and a client, you can run either `osconfig server` or `osconfig rawfs` on that host. Either command line will configure the host to be both an ObjectStore server and client.

After `osconfig` configures ObjectStore, it verifies the configuration by checking the following:

- The server is running and is addressable by the client.
- The cache manager is available as `setuid root`.
- The library schemas are available.

For more information about using `osconfig` when installing ObjectStore, refer to the *Installation Guide for UNIX*.

The `osconfig` utility can also be used to enable ObjectStore to use failover — either ObjectStore- or cluster-managed failover. For more information, see Failover on page 285.

Performing ObjectStore Upgrades

This section describes how to use the `osconfig` utility to upgrade an existing server with a new release of ObjectStore. Before running the `osconfig` utility, however, you must install the new version of ObjectStore in a *new* directory and change the `OS_ROOTDIR` environment variable to point to the new directory. After you have installed ObjectStore and used `osconfig` to configure the upgraded server, you are free to delete the old `OS_ROOTDIR` directory and its contents. For information installing ObjectStore, see the Install Guide that accompanies the installation package. For more information about the `OS_ROOTDIR` variable, see `OS_ROOTDIR`.

To use the `osconfig` utility to upgrade an existing ObjectStore server with an updated version of ObjectStore, use either of the following command lines.

```
osconfig server -upgrade
```

```
osconfig rawfs -upgrade
```

The `osconfig` utility will prompt you for configuration information. The `osconfig` utility will shutdown the old server so that you can install the updated version of

ObjectStore. After the installation is complete, `osconfig` will restart the new server with the updated configuration information. No server parameters will be modified. You must restart the clients.

Rolling upgrade

You can also use the `osconfig` utility to perform rolling upgrades. A *rolling upgrade* enables you to upgrade an ObjectStore failover server without stopping client use of the server. Initially, `osconfig` shuts down the primary server, causing it to failover to the secondary server and allowing the primary server to be upgraded. During the upgrade process, the secondary server takes over the services of the primary server. After the primary server has been upgraded and is ready to be brought back online, `osconfig` shuts down the secondary server, causing it to failover to the primary server. At this point, the primary server resumes services and the secondary server can be upgraded.

To use the rolling upgrade feature, you must have configured ObjectStore to use either ObjectStore-managed failover or cluster-managed failover; see Failover on page 285. To perform a rolling upgrade on a system configured with ObjectStore-managed failover, invoke the following command line:

```
osconfig failover [-server server1 server2] -upgrade
```

If you are using cluster-managed failover, invoke the following command line:

```
osconfig cluster -logical_host list_of_names -upgrade
```

After either command line is invoked, `osconfig` will prompt you to enter new configuration information. The rawfs and transaction log information will remain the same. The `Identical Pathnames on Failover Server` server parameter will be re-generated based on the configuration information you entered. (For more information about this server parameter, see Identical Pathnames on Failover Server on page 90.) All other server parameters will remain the same.

Note that during the rolling upgrade, the server will failover several times in the process of being upgraded. Its clients will pause and retry each time failover occurs.

If the server is not running at the time you invoke `osconfig -upgrade`, the server will be upgraded without the rolling upgrade feature.

oscopy

The oscopy utility makes a copy of an ObjectStore database. A key benefit of oscopy is that it performs transaction-consistent database copying without incurring locking conflicts.

The oscopy utility cannot copy segment-level permissions and it cannot copy ObjectStore/Single databases. For copying ObjectStore/Single databases, use the standard operating system copy facilities. When copying an ObjectStore/Single database, make sure that the transaction log has been propagated and that no application has the database open.

Syntax

This utility has three forms. Note that the second form includes an option.

```
oscopy [-R] source target
oscopy database1 . . . databaseN rawfs_dir
```

Arguments

source

Specifies the ObjectStore file or rawfs database to be copied.

target

Specifies the path for the copy. ObjectStore either creates this database or overwrites it.

source_dir

Specifies the path of the rawfs directory to be copied.

target_dir

Specifies the target rawfs path name. If this directory does not exist, ObjectStore creates it if its base name exists. ObjectStore copies the source directory recursively into the target directory.

*database1, database2, database*n

Specifies the ObjectStore file or rawfs database or databases to be copied.

rawfs_dir

Specifies the rawfs directory to which the databases are copied.

Options

-R

Instructs oscopy to copy a directory recursively. You must specify a rawfs directory for both the source and the destination path names. The top-level name of the destination path name must exist before you issue oscopy.

Description

The oscopy utility has three forms. The first copies a file or rawfs database to another file or rawfs database. The second recursively copies a rawfs directory and its contents to another location. The third copies a database or databases to a rawfs file

system. In this form, you can specify either file or rawfs databases as sources; the target must always be a rawfs directory.

You cannot specify wildcards in database path names.

When you specify more than one database as a copy source, `oscopy` ensures transaction consistency among the specified databases for a particular moment in time.

Copying a rawfs database to a file database results in the loss of segment-level access control information (`os_segment_access`).

If a failover occurs while `oscopy` is copying a database, the utility restarts from the beginning.

Your database might appear to have a different size after you use `oscopy` to copy it. This is because the server might allocate the copy in a way that is different from the way it allocated the original database. Also, when you perform `oscopy`, the size of the database is set. The server can make exactly the right amount of space available for the copy of the database. For example, using `oscopy` after running `oscompact` eliminates the free space reclaimed by compaction, reducing the size of the file database.

There are many conditions that can affect how `oscopy` interprets path names:

- Settings of environment variables
- Whether there is a locator file
- Whether file systems are NFS-mounted
- Symbolic links

When you copy a file and the result is not what you expect, be sure to consider these conditions.

API

See `os_dbutil::copy_database()` on page 181 in the *C++ API Reference*.

osdbcontrol

The `osdbcontrol` utility allows a user to perform various control options on a list of databases, primarily to move them offline or back online.

Syntax

```
osdbcontrol options pathname ...
```

Argument *pathname*

Specifies the path name of a database that is the target of the `osdbcontrol` command. Multiple path names can be listed.

Options `-cluster_size seg_num cluster_num n_bytes`

Sets the size in bytes of the specified cluster. The cluster must be identified by segment number, cluster number, and the pathname of the database. The size (*n_bytes*) is specified as a decimal integer, which may be followed by the optional suffix `K` (kilobytes), `M` (megabytes), or `G` (gigabytes). Multiple database pathnames can be specified and the command is applied individually to the specified cluster in each database.

The following command line sets the size of segment 2, cluster 8 to 14 kilobytes:

```
osdbcontrol -cluster_size 2 8 14K \temp\test1.db
```

This option enables the user to presize a cluster in order to minimize fragmentation. You can also presize a cluster programmatically, as described in `os_cluster::set_size()` in Chapter 2, Class Library of the *C++ API Reference*. Note that you cannot make a cluster smaller than needed to hold all of the objects in it.

For more information about fragmentation, see Managing Database Fragmentation on page 44.

`-size n_bytes`

Sets the size in bytes of the specified database file. The size (*n_bytes*) is specified as a decimal integer, which may be followed by the optional suffix `K` (kilobytes), `M` (megabytes), `G` (gigabytes), or `T` (terabytes). Multiple database pathnames can be specified and the command is applied individually to each database.

The following command line sets the size of the specified database file to 100 megabytes:

```
osdbcontrol -size 100M \temp\test1.db
```

This option enables the user to presize a database file in order to minimize fragmentation. You can also presize a database file programmatically, as described in `os_database::set_size()` and `os_database::set_size_in_sectors()` in Chapter 2, Class Library of the *C++ API Reference*. Note that you cannot decrease the size of a database file.

For more information about fragmentation, see Managing Database Fragmentation on page 44.

`-offline`

Sets a database offline after all users have closed the database specified by *`pathname`* and after all data has been propagated to the database.

`-reason string`

Only allowed with `-offline`. The string specified is appended to the offline exception message.

`-kill_clients`

Only allowed with `-offline`. This option disconnects all clients accessing the database before putting the database offline.

When using the `-kill_clients` option, if the server's authentication is set to something other than `NONE` (authentication is `SYS` by default), any user can disconnect clients that the user owns. Otherwise, authentication fails and no clients are disconnected.

When the Admin User is set, that user can disconnect all clients. Any user can disconnect all clients if the authentication is set to `NONE`.

Without the `-kill_clients` option, the offline request first waits for all clients to close the database and then puts the database offline.

`-online`

Puts the database specified by *`pathname`* back online.

`-status`

Returns the current status of the specified database. For an online database, the number of opens on the database along with a list of clients currently accessing the database are displayed.

`-statistics`

Displays various statistics about the database.

If you use an invalid combination of options or if no options are specified, an `Invalid combination of options` exception is thrown.

Note: If a database is set offline while `osbackup` is backing up that database, the `osbackup` process will ignore the offline state and complete the backup. Using the `-kill_clients` option while `osbackup` is backing up that database will disconnect the `osbackup` client, leaving the backup logs in an inconsistent state.

API

See `os_dbutil::osdbcontrol()` on page 184 in the *C++ API Reference*.

Examples

```
osdbcontrol -offline -kill_clients -reason "will be back in 5 minutes" parts.db
```

In the above example the `-kill_clients` option first disconnects all clients accessing the database and then puts the database offline.

```
osdbcontrol -online parts.db
```

The above example places the database back online.

```
osdbcontrol -status parts.db
```

In the above example, the `-status` option outputs the current status of the database, as shown below. The output shows that the database is online and four clients currently have the database open. If the database is going offline, then the status output will show `OSDBCONTROL_DATABASE_IS_GOING_OFFLINE` and list the clients that still have the database open.

```
osdbcontrol for database: parts.db
Status: OSDBCONTROL_DATABASE_IS_ONLINE
Number of clients with database open = 4
  Client #64 (mangle/16197/partapp -db parts.db/0)
  priority=0x8000, duration=1 seconds, work=0, no transaction
    on server
  Client #65 (mangle/16197/partapp -db parts.db/1)
  priority=0x8000, duration=1 seconds, work=0, no transaction
    on server
  Client #66 (mangle/16197/scanner -db parts.db/2)
  priority=0x8000, duration=0 seconds, work=0, no transaction
    on server
  Client #67 (mangle/16197/partapp -db parts.db/3)
  priority=0x8000, duration=0 seconds, work=20, transaction in
    progress
```

osdf

The `osdf` utility shows the amount of used and available disk space for the rawfs on the specified server.

Syntax

`osdf` *`hostname`*

Argument *`hostname`*

Specifies the name of the host for which you want to display rawfs disk space information.

Description

If one or more of the partitions that make up the rawfs are expandable and there is free disk space in the file system holding such a partition, the rawfs grows as needed. In this situation, the `osdf` utility does not show the amount of growth room available.

API See `os_dbutil::disk_free()` on page 181 in the *C++ API Reference*.

Examples

`osdf elvis`

```
Filesystem   kbytes     use        avail      capacity
elvis        95749      533        95215      0%
```

osdump

The `osdump` utility dumps an ObjectStore C++ database or group of databases to ASCII files.

Note: if you are dumping an ObjectStore Java database, use the `osjidump` utility. See osjidump on page 186.

Syntax

The syntax for `osdump` is

`osdump [`*`options`*`] `*`pathname ...`*

Argument

`pathname ...`

One or more database path names, separated by spaces, specifying the database or databases to be dumped.

Options

`-d |-debug `*`level`*

Sets debugging level to the specified value *`level`*. The range is `1`–`6`. The default is that debugging information is not displayed.

`-dir | -dest_dir `*`dump_pathname`*

Specifies the path for the dump output.

`-pr | -process `*`processes`*

Specifies the number of processes available for multiprocessing. The default is `1`.

Description

Each execution of the `osdump` utility does the following:

- Creates a group of ASCII files with names in the form *`databasename_segmentN.dmp`*. A separate file is created for each segment in each database. These files represent the data content of the database in ASCII format and are used by the `osload` application you create.
- Creates a group of ASCII files with names in the form *`databasename_segmentN.fix`*. A separate file is created for each segment in each database. These files contain information about collections in the databases.
- Creates a table file named `db_table.dmp`. This file contains a representation of the physical layout of the dumped databases and is used by the `osload` application you create.

  See Default Dumper ASCII Format for C++ Databases on page 175 for a description of the layout of this file.

- Creates a file named *`databasename.scm`*. This file contains a representation of the database schema and is used by the `osload` application you create.

These ASCII files are used as input files to the `osload` utility. The dumped files must be available to the `osload` utility when the utility runs. After the `osload` utility successfully runs, you can delete the files generated by the `osdump` utility.

See also osload on page 192.

Default vs. customized dump and load

For the C++ interface, you can use the default dump and load processes or customize the dump and load of particular types of objects. Customization is appropriate for certain location-dependent structures, such as hash tables. Databases with unions, pointers-to-members, or address-dependent data of types that are not C++ pointers or ObjectStore references or soft pointers cannot be dumped by this utility and need customized dump and load applications.

Upgrading a Database

To determine when to customize, see When Is Customization Required for C++ Databases? on page 174 To learn how to customize, see Chapter 6, Dump/Load Facility, in the *Advanced C++ API User Guide*.

For a description of how to use `osdump` and `osload` to migrate databases from Release 5.1 format to Release 6.1 format, see Upgrading a C++ Database on page 54.

Considerations

When you are planning how to dump and load a database, the following points might influence your plans:

- Any database that you dump must be taken offline.
- Dumping and loading a large database with segments or collections comparable in size to the available address space might not be possible.
- Indexes for collections are always put into the same segment as the collection even if the original index was in a different segment.
- If you are dumping a database with cross-database pointers or references to objects in other databases, you must dump the other affiliated databases at the same time; otherwise, the integrity of the database you are dumping will not be maintained.
- User-defined constructors for nonspecialized classes are not executed during the load.
- If you plan to load a database on a platform with a different architecture from the architecture of the original database, you should neutralize the database and emit the loader on the platform with the target architecture.

When Is Customization Required for C++ Databases?

In most cases, customization is not required. If you have a database with objects whose structure depends on the locations of other objects, you might have to customize the dumping and loading of those objects.

A dumped object and its equivalent loaded object do not necessarily have the same location; that is, the same offsets in their cluster. Among the implications of this are the following:

- Other objects might use different pseudoaddresses to refer to them.

- Their addresses might hash to different values; that is, `objectstore::get_pointer_numbers()`, for example, might return different values for them.

The default dumper and loader take into account the first implication, and the loader adjusts all pointers automatically in loaded databases to use the new locations.

The default dumper and loader also take into account the second implication for ObjectStore types including collections with hash-table representations. Because a dumped collection element hashes to a different value than the corresponding loaded element does, their hash-table slots are different. The facility, then, does not simply dump and load the array of slots based on fundamental values (which would result in using the same slot for the dumped and loaded objects).

Instead, it dumps the collection in terms of sequences of high-level API operations (that is, string representations of `create` and `insert` arguments) that the loader can use to recreate the collection with the appropriate membership.

The default dumper and loader do not take into account the second implication for non-ObjectStore classes. If you have your own classes that use hash-table representations, you must customize their dumping and loading. Any other location-dependent details of data structures (such as encoded offsets) also should be dealt with through customization.

See Chapter 6, Dump/Load Facility, in the *Advanced C++ API User Guide*.

Default Dumper ASCII Format for C++ Databases

Each `db_table.dmp` file has the format for `database_table` described in the following section.

For each dumped database, *`filename`*`.db`, the files *`databasename_segmentN`*`.dmp` have the format shown for `database`, in the following information:

```
meta::=
  <
    objectstore major release number
    objectstore minor release number
    objectstore maintenance release number
    name of the dump format
    dump format version number
    build number
    value of the application architecture
  >

database_table ::=
  databases [number_databases]
    {database_entry [database_entry]*}

number_databases ::=
 the integer number of dumped databases

database_entry ::=
  <
    pathname
    database_size
    number_segments
    odi_release
    architecture (date)
```

```
  >

database ::=
  database [database_index] pathname segments roots
  finalization fixups

pathname ::=
 the pathname of the database being dumped

database_size ::=
 the size of the database in an integral number of bytes

number_segments ::=
 the integral number of segments contained in the database

odi_release ::=

 the ObjectStore release information

architecture ::=
 the host architecture set for the database

date ::=
 the date the database was last modified

database_index ::=
 the index of this database within the list of databases being
 dumped (0-based)
roots ::=
  roots [number_roots]  {root [, root]*}

root ::=
  name (Type) id

segments ::=
  segments [segment]*

segment ::=
  segment segment_number [segment_size]
    (pathname) data

segment_number ::=
 integral segment number of this segment within its database

data ::=
 (objects)*

cluster ::=
  cluster [cluster_size] {objects}

objects ::=
  object*

object ::=
  id (type) value

id ::=
  <database_index,segment_number,cluster_number,offset>

offset ::=
 integral value denoting the byte offset of an object within its cluster

type ::=
  integral | real | pointer | reference | array | class

value ::=
  character | integral | floating_point | pointer_value | reference |
    collection | string | array_elements | class_members
```

```
integral ::=
  char | signed char |unsigned char| signed short|
    unsigned short| int| unsigned int | signed long |
      unsigned long
real ::=
  float |double

pointer ::=
  type*

reference ::=
  type&

array ::=
  array type [ size ]

class ::=
 (class| struct| union) name

character ::=
 ‘ c' where c is any printable ascii character or escaped hex value

integral ::=
 any non-floating-point decimal number

floating_point ::=
 any floating-point decimal number

pointer_value ::=
  id

string ::=
 "s" where s is any sequence of printable ascii characters or
 escaped hex value with "" escaped as "\"" and '\' escaped as "\\".

array_elements ::=
  {value [, value]*}

class_members ::=
  {value [, value]*}
```

Each object is emitted as a single line of text.

The special storage types `cluster`, `segment`, and `database` denote underlying ObjectStore storage structures. When a storage type appears, each object following is contained within that storage structure.

Other types denote C++ type constructs. Values appear as single values or as a bracketed comma-separated list of values. Base class instances and other embedded subobjects are flattened into a `class_members` list.

ObjectStore references or soft pointers, collections, cursors, indexes, and queries are instances of Object Store types that require special treatment. The following special dump formats are used for them:

```
reference ::=
  id

collection ::=
  simple_collection |dummy_cursor |cursor | cursor_with_index |
      cursor_with_range | collection_index |
        collection_element_load|collection_query

simple_collection::=
  [behavior cardinality collection_type representation_enum]
```

```
behavior::=
  integral

cardinality::=
  integral

collection_type::=
  string

representation_enum::=
  integral

dummy_cursor::=
  [D]

cursor::=
  [C options]

collection_reference::=
  reference

safe_flag::=
  integral

cursor_with_index::=
  [C options

    {I path_name_length path_name element_name_length
      element_name}]

path_name_length::=
  integral

path_name ::=
  string

element_name_length::=
  integral

element_name::=
  string

cursor_with_range::=
  [C collection_reference safe_flag
    {R range_type key_type low_condition low_value H
      high_condition high_value}]

range_type::=
  integral

key_type::=
  integral

low_condition::=
  boolean

low_value::=
  integral

high_condition::=
  boolean

high_value::=
  integral

boolean::=
  0 | 1

collection_index::=
```

```
  [[{path_name_length path_name element_type_length
    element_type_name}]*]

collection_element_load::=
  [[element_reference]*]

element_reference::=
  reference

element_type_length::=
  integral

element_type_name::=
  string

collection_query::=
  [element_type_length element_type  <query_string_length
    query_string> <file_name_length file_name>
      <line_number>]

query_string_length::=
  integral

query_string::=
  string

file_name_length::=
  integral

file_name::=
  string

line_number::=
  integral
```

osexschm

The `osexschm` utility lists the names of all classes in the schema referenced by the specified database.

Syntax

```
osexschm [-detail] pathname
```

Argument

`pathname`

Specifies a file or rawfs database.

Option

`-detail`

Describes the structure of every class in detail.

Description

For each class, `osexschm` indicates whether an object of the class type can be allocated persistently.

Schema protection

If you are developing an application and will be running this utility on a protected schema database, ensure that the correct key is specified for the environment variables `OS_SCHEMA_KEY_LOW` and `OS_SCHEMA_KEY_HIGH`. If the correct key is not specified for these variables, the utility fails. ObjectStore signals

```
err_schema_key _CT_invalid_schema_key,
"<err-0025-0151> The schema is protected and the key provided did not
match the one in the schema."
```

osgc

The `osgc` utility frees storage associated with persistent objects that are unreachable.

The command-line utility for collecting garbage is `osgc`.

Syntax

`osgc [`*`options`*`] `*`database_name`*

Argument

`database_name`

Specifies the pathname for the database on which to run `osgc`.

The `osgc` utility is *only* installed when you install the ObjectStore Java interface. For the ObjectStore Java interface, executing `osgc db` is the same as calling the `Database.GC()` method on the `db` database.

Options

`-seg` *`segment_id`*

Collects garbage from only the specified segment. The segment must have the segment reference policy `objectstore::export_id_access_required`. By default, the `osgc` utility operates on the entire database.

`-retries` *`number`*

Indicates the number of times the tool tries to resume the sweep phase of garbage collection after it waits for a lock. The default is `10`.

For the ObjectStore Java interface, this option is identical to the `com.odi.gc.retries` property.

`-retryInterval` *`interval`*

Indicates the number of milliseconds the sweep operation waits between sweep attempts for a concurrency conflict to be resolved before it tries to resume the sweep. The default is `1000`.

For the ObjectStore Java interface, this option is identical to the `com.odi.gc.retryInterval` property.

`-lockTimeOut` *`interval`*

Indicates the number of milliseconds the sweep operation waits for a lock conflict to be resolved. If it is not resolved in the specified length of time, the tool aborts the current transaction and starts a new transaction. ObjectStore rounds this value up to the nearest second. The default is `1000`.

For the ObjectStore Java interface, this option is identical to the `com.odi.gc.lockTimeOut` property.

`-transactionPriority` *`n`*

Specifies the transaction priority associated with transactions started by the tool. The server uses this specification when it must determine the transaction that must be the victim in a deadlock. This number is intentionally low so that the garbage collection transaction is the deadlock victim of choice. The default is `0`.

For the ObjectStore Java interface, this option is identical to the `com.odi.gc.transactionPriority` property.

`-displayGarbage` *`level`*

Displays information about the candidates for garbage collection instead of actually destroying the candidates. The level you specify determines the amount of information the tool displays. Level 1 lists the number of objects per segment that would be destroyed. Level 2 is not currently supported. Level 3 lists the location of each garbage collector candidate. Level 4 lists the roots of garbage graphs. Level 4 can require intensive computations.

`-statistics`

Displays statistics for the garbage collection operation. These include the total number of reachable objects and the total number of garbage objects.

Description

Performing garbage collection in a database

The ObjectStore persistent garbage collector (`osgc`) deletes unreferenced, non-exported objects and ObjectStore collections in an ObjectStore database. Applications can continue to use a database while garbage collection is in process. Note, however, that you should *not* retain transient references to non-exported objects while you are using `osgc`.

Persistent garbage collection frees storage associated with objects that are unreachable. It does not move remaining objects to coalesce the free space. (See oscompact on page 160.)

The `osgc` utility performs its job in two major phases. In the mark phase, `osgc` identifies the unreachable objects. In the sweep phase, `osgc` frees the storage used by the unreachable objects.

A segment is the smallest storage unit that can be garbage collected. You can specify a segment or a database for garbage collection. You can only specify a segment if it has the segment reference policy `objectstore::export_id_access_required`.

C++ usage note

Normally, databases resulting from ObjectStore applications written in C++ do not require garbage collection because all storage allocation is handled explicitly.

`osgc` can be useful as a debugging tool. For example, if unreferenced objects are being harvested, it is an indication of a persistent memory leak. The identity of these objects can be a clue to the root of the problem.

Caution

If your application relies on interdatabase pointers, you should use segments with the segment reference policy of `objectstore::export_id_access_required`. Otherwise, references to one database from another will not be detected by `osgc` and objects pointed to by references from other databases will be seen as unreferenced and, therefore, removed.

Applications can continue to use a database while persistent garbage collection is in progress — with the restriction noted in the next paragraph. Garbage collection locks portions of a segment as needed, as if it were just another application. In this way, the `osgc` utility minimizes the number of pages that are locked and the duration for

which the locks are held. Also, the `osgc` utility retries operations when it detects lock conflicts.

Restriction

Applications must not retain references to non-exported objects in a database while it is being processed by the `osgc` utility. This restriction applies to both the C++ and Java interfaces to ObjectStore.

By default, the `osgc` utility runs with a transaction priority of `0`. Consequently, it is the preferred victim when the server must terminate a transaction to resolve a deadlock. At a later time, the `osgc` utility redoes the work that was lost when the transaction was aborted.

The GC uses read- and write-lock timeouts of short duration. This avoids competition with other processes for locks. If the `osgc` utility cannot acquire a lock because of a timeout, it retries the operation at a later time.

You must not simultaneously run multiple `osgc` processes against the same database or segment. You must synchronize `osgc` processes so that a new `osgc` process on a database or segment is initiated only after the previous `osgc` process on the database or segment has run to completion.

osglob

The `osglob` utility performs ObjectStore file name expansion.

Syntax

`osglob` *`wordlist`*

Argument *`wordlist`*

Specifies strings, such as rawfs path names, containing wildcards that you want to expand into all matching path names.

Description

The `osglob` utility can perform wildcard processing similar to regular expression wildcards `*`, `?`, `{}`, and `[]`.

UNIX On UNIX systems, when you are operating on a rawfs database, you must enclose the wildcard in quotation marks ("") or precede it with a backslash (\) to keep the shell from interpreting a wildcard as a shell wildcard.

API See `os_dbutil::expand_global()` on page 181 in the *C++ API Reference*.

oshostof

The `oshostof` utility displays the host name of the specified database to standard output.

Syntax

`oshostof` *`pathname`*

Argument *`pathname`*

Specifies the database for which you want to display the host name.

Description

The `oshostof` utility can operate on file or rawfs databases.

Normal path name syntax is supported.

When you specify a path that is a symbolic link, `oshostof` displays the host name of the database that the link points to.

When you specify the path of a server-remote database, the `oshostof` utility returns the name of the host where the database resides.

API See `os_dbutil::hostof()` on page 182 in the *C++ API Reference*.

Example

A typical use is as follows:

```
ossvrchkpt 'oshostof a/b/c'
```

osjidump

The `osjidump` utility dumps an ObjectStore Java database or group of databases to ASCII files.

The `osjidump` utility is only installed when you install the ObjectStore Java interface. The utility is located in the `OSJI/bin` directory.

Syntax

The syntax for `osjidump` is

```
osjidump [options] pathname ...
```

Argument

`pathname ...`

One or more database path names, separated by spaces, specifying the database or databases to be dumped.

Options

`-d |-debug` *`level`*

Sets debugging level to the specified value *`level`*. The range is `1`–`6`. The default is that debugging information is not displayed.

`-dir | -dest_dir` *`dump_pathname`*

Specifies the path for the dump output.

`-pr | -process` *`processes`*

Specifies the number of processes available for multiprocessing. The default is `1`.

Description

Each execution of the `osjidump` utility does the following:

- Creates a group of ASCII files with names in the form *`databasename_segmentN.dmp`*. A separate file is created for each segment in each database. These files represent the data content of the database in ASCII format and are used by the `osjiload` utility.
- Creates a group of ASCII files with names in the form *`databasename_segmentN.fix`*. A separate file is created for each segment in each database. These files contain information about collections in the databases.
- Creates a table file named `db_table.dmp`. This file contains a representation of the physical layout of the dumped databases and is used by the `osload` application you create.
- Creates a file named *`databasename.scm`*. This file contains a representation of the database schema and is used by the `osjiload` utility

These ASCII files are used as input files to the `osjiload` utility. The dumped files must be available to the `osjiload` utility when it runs. After the `osjiload` utility successfully runs, you can delete the files generated by the `osjidump` utility.

See also osjiload on page 188.

Upgrading a Database

For a description of how to migrate databases from Releases 3.0 format to Release 6.3 format, see Upgrading a Java Database on page 56 (note, for Release 3.0, you will use a version of the `osdump` utility).

Considerations

When you are planning how to dump and load a database, the following points might influence your plans:

- Any database that you dump must be taken offline.
- Dumping and loading a large database with segments or collections comparable in size to the available address space might not be possible.
- Indexes for collections are always put into the same segment as the collection even if the original index was in a different segment.
- If you are dumping a database with cross-database pointers or references to objects in other databases, you must dump the other affiliated databases at the same time; otherwise, the integrity of the database you are dumping will not be maintained.

osjiload

The utility `osjiload` creates an ObjectStore Java database or group of databases using as input the ASCII files generated by the `osjidump` utility. The resulting databases are equivalent to the ones from which the ASCII files were produced.

The `osjiload` utility is only installed when you install the ObjectStore Java interface. The utility is located in the `OSJI/bin` directory.

Syntax

The syntax for `osjiload` is

`osjiload [`*`options`*`]`

Options

`-cwd | -current_dir`

Tells `osjiload` to recreate databases in the current working directory rather than in their original location.

`-d | -debug` *`level`*

Sets debugging level to the specified value *`level`* if debugging information is available. The range is `1`–`6`. The default is that debugging information is not displayed.

`-dir | -source_dir` *`pathname`*

Specifies the directory from which to read the input files such as `db_table.dmp`.

`-pr | -process` *`processes`*

Specifies the number of processes available for multiprocessing. The default is `1`.

`-rc -recluster` *`size_in_bytes`*

Specifies the maximum reclustering size to a given integral value in bytes. By default, the original cluster size is retained.

`-wdir |-workdir` *`location`*

Specifies the location to use for temporary work storage.

Description

For given ASCII input, the databases created by the `osjiload` utility have the same file names as the databases from which the ASCII was generated (as stored in `db_table.dmp`).

-cwd option

The `-cwd` option forces `osjiload` to ignore paths from `db_table.dmp` and create the databases in the current working directory.

If you do not set the `-cwd` option, the databases have the same path names as stored in `db_table.dmp`. If files with the given paths already exist (for example, because the dumped databases are still in their original locations), the `osjiload` utility aborts. This prevents you from overwriting your original databases.

To load a database or group of databases from a set of `osjidump`-generated ASCII files, run the `osjiload` utility against the dumped ASCII files.

CLASSPATH

The `osjiload` utility requires the `CLASSPATH` environment variable include the OSJI classes `OSJI/osji.jar` and `OSJI/tools.jar` as well as the postprocessed class files used in your database schema.

Upgrading a Database

For a description of how to use `osjidump` and `osjiload` to migrate databases from Release 3.0 format to Release 6.1 format, see Upgrading a Java Database on page 56.

osln

The `osln` utility creates a symbolic link in the rawfs hierarchy.

Syntax

`osln` *`pathname linkname`*

Arguments

`pathname`

The path name of the rawfs directory or database that you want to point to.

`linkname`

The path name of the rawfs directory or database that is the new link. It points to *`pathname`*.

Description

Different links can point to the same rawfs path name.

To indicate hosts, specify path names in the form

`host``::/`*`pathname`*

Limitation

To access a particular database or directory, a client can follow as many as 15 cross-server links. For example, a client traverses a link to Server Q. Server Q sends the client to Server P. Server P sends the client to another Server or even back to Server Q. Each connection to a server counts as one link. It does not matter whether the server was previously connected to the chain. When the client reaches the sixteenth link, ObjectStore signals the error message `err_too_many_cross_svr_links`.

To access a particular database or directory in its rawfs, the server can traverse as many as ten same-server links. When the server reaches the eleventh link, ObjectStore signals the error message `err_too_many_links`.

In a chain of links, a client can return to a server that it contacted earlier in the chain. In this situation, the server's count of links within its rawfs begins with 1. It does not continue the count from where it left off during the previous connection. Each time a link sends the client to a server, the server can follow as many as ten links within its rawfs.

These limits allow ObjectStore to catch circular links such as the following: A is a link to B, and B is either directly or indirectly a link to A.

When links are needed

Links within the rawfs are useful in many situations, including the following:

- You move a database. You can define a link so that programmers need not modify applications that use the moved database.
- You want to store all databases in one place but allow applications to refer to the databases in different ways.
- You want to set up applications so they always refer to one location, and that location is a link to the actual database.

Removing a link

To remove a link, use the `osrm` utility. The syntax is `osrm` *`linkname`*.

See osrm on page 213.

API

See `os_dbutil::make_link()` on page 183 in the *C++ API Reference*.

Examples

In the following example, `link_to_db` in `canard`'s rawfs points to `real_db` in `web-foot`'s rawfs:

```
osln web-foot::/real_db canard::/link_to_db
```

In the next example, `link_to_db` points to `real_db`, and both databases are in the same rawfs:

```
osln web-foot::/real_db web-foot::/link_to_db
```

osload

The `osload` utilitycreates an ObjectStore C++ database or group of databases using as input the ASCII files generated by the `osdump` utility. The resulting databases are equivalent to the ones from which the ASCII files were produced.

You use the `osload` utility supplied by the ObjectStore installation to create the source files for an `osload` application that is tailored specifically to load databases you have dumped with the `osdump` utility.

Syntax

The syntax for `osload` is

```
osload [options]
```

Options

`-cwd | -current_dir`

Tells `osload` to recreate databases in the current working directory rather than in their original location.

`-d | -debug level`

Sets debugging level to the specified value *`level`* if debugging information is available. The range is `1`–`6`. The default is that debugging information is not displayed.

`-dir | -source_dir pathname`

Specifies the directory from which to read the input files such as `db_table.dmp`.

`-e | -emit pathname`

Tells `osload` to generate the source code for a loader executable for the application schema specified by *`pathname`*.

`-pr | -process processes`

Specifies the number of processes available for multiprocessing. The default is `1`.

`-rc -recluster size_in_bytes`

Specifies the maximum reclustering size to a given integral value in bytes. By default, the original cluster size is retained.

`-wdir |-workdir location`

Specifies the location to use for temporary work storage.

Description

For given ASCII input, the databases created by the `osload` utility have the same file names as the databases from which the ASCII was generated (as stored in `db_table.dmp`).

-cwd option

The `-cwd` option forces `osload` to ignore paths from `db_table.dmp` and create the databases in the current working directory.

If you do not set the `-cwd` option, the databases have the same path names as stored in `db_table.dmp`. If files with the given paths already exist (for example, because the dumped databases are still in their original locations), the `osload` utility aborts. This prevents you from overwriting your original databases.

To load a database or group of databases from a set of `osdump`-generated ASCII files, generate the source code for a loader application by running `osload` with the `-emit` option against the application schema, build the loader application, and then run the loader application against the ASCII files.

On UNIX, use the `make` utility and the generated makefile `makefile.unx`. On Windows, use `nmake` and the generated makefile `makefile.w32`. The makefiles are generated along with the loader source code when you run `osload` with the `-emit` option

Upgrading a Database

For a description of how to use `osdump` and `osload` to migrate databases from Release 5.1 format to Release 6.1 format, see Upgrading a C++ Database on page 54.

osls

The `osls` utility lists the contents of the specified directory.

Syntax

```
osls [-dlRsu] pathname ...
```

Argument

`pathname ...`

Specifies one or more rawfs or native file directories for which you want to list the contents.

Options

`-d`

Lists the information about the directory itself, rather than the contents. This option operates on rawfs directories only.

`-l`

Displays information about directory contents in long format, including the size in bytes.

`-R`

Recursively lists the contents of the specified directory.

`-s`

Causes the size to be displayed in 1 KB blocks. This option operates on rawfs directories only.

`-u`

Lists the user name of the owner of the contained databases. This option operates on rawfs directories only.

Description

When a path name includes links, ObjectStore identifies the path name as the path name to which the symbolic link chain points. This is true even if an alternative name was specified at creation.

The `osls` utility ignores trailing and multiple slashes in path names. It accepts a combination of rawfs path names and file path names. When you specify a local directory, you cannot specify a remote file-server host in the path name of the local directory. The `osls` utility passes the operation to a local native utility. If you specify a remote file-server host name, ObjectStore informs you that you specified an illegal path name.

Wildcards

This utility can perform wildcard processing using regular expression wildcards `*`, `?`, `{}`, and `[]`.

UNIX

On UNIX, when you are operating on a rawfs database, you must enclose the wildcard in quotation marks (" ") or precede it with a backslash (\) to keep the shell from interpreting the wildcard as a shell wildcard.

API

See `os_dbutil::list_directory()` on page 183 in the *C++ API Reference*.

osmkdir

The `osmkdir` utility creates a directory in the rawfs.

Syntax

```
osmkdir [-p] [-m octal_mode] directory
```

Argument

`directory`

Specifies a rawfs directory path name.

Options

`-m` *`octal_mode`*

Indicates that the new directory has the permission mode as specified by *`octal_mode`*. Specify the protection mode that you want the directory to have. The default mode is 0700.

`-p`

Specifies that ObjectStore should create any missing directories that are needed to make the specified directory path exist.

Description

Use `osmkdir` to create a rawfs directory. You also can use `osmkdir` to create a nonrawfs directory. When you create a nonrawfs directory, you cannot specify a remote file-server host in the path name of the nonrawfs directory. The `osmkdir` utility passes the operation to a local native utility. If you specify a remote file-server host name, ObjectStore informs you that you specified an illegal path name. If you specify the `-p` option, it works if the native utility supports that feature.

API

See `os_dbutil::mkdir()` on page 184 in the *C++ API Reference*.

osmv

The `osmv` utility moves a database, directory, or link.

Syntax

The `osmv` utility has two forms that apply to rawfs databases and a third form that applies to file databases.

For rawfs:

`osmv [-fi] path1 path2`

`osmv [-fi] path1... pathn dir`

For file databases:

`osmv [-fi] path1 path2`

Arguments

path1
path1 ... pathn

Specifies the path of a file database or a rawfs database, link, or directory that you want to move.

path2

Specifies the new path for the file database or rawfs database, link, or directory. If *path2* is a link to a directory, ObjectStore places *path1* in the pointed-to directory.

dir

Specifies a rawfs directory into which you want to move the specified rawfs databases, links, or directories.

Options

`-f`

Forces execution. Errors are not reported.

`-i`

Specifies interactive mode. ObjectStore prompts you to confirm for each specified database that you really want to move it.

Description

The `osmv` utility moves rawfs databases, directories, and links within a rawfs or from one rawfs to another. It also moves file databases within the file system. A side effect of `osmv` is to rename a file or directory.

As shown in the Syntax section, there are three forms of the command line for the `osmv` utility.

In the first form, if *path1* and *path2* are rawfs databases or links, `osmv` moves (changes the name of) *path1* to *path2*. If *path2* already exists, the utility removes it, then moves *path1* to *path2*. If *path1* is a rawfs directory, then *path2* must not already exist. `osmv` moves (changes the name of) the *path1* directory to the *path2* directory.

In the second form, `osmv` moves one or more databases, links, or directories into the last directory in the list. The utility maintains the original names of the moved entities. The directory into which you are moving items must already exist and you must have write permission for that directory.

In the third form, `osmv` moves (changes the name of) file database *`path1`* to file database *`path2`*.

Procedure

When moving rawfs databases to another server, the `osmv` utility moves an item by doing the following:

1. Removes the destination, if it exists
2. Copies the source to the destination
3. Removes the source

This allows for consistent databases in the event of a server crash. If the server crashes during the `osmv` operation, there might not be a destination database, but there would always be a source database. When moving rawfs paths on the same server, ObjectStore directly renames the item.

External pointers and references

After you move a database, you need to use the `osaffiliate` utility to fix external pointers and references. See osaffiliate on page 135.

Native move commands

While you can use native move commands to move ObjectStore file databases, you forfeit the database consistency protection that `osmv` provides. If the server crashes before propagating all changes to the database, the server cannot find the changes at recovery time and the database is corrupted.

When you specify a file database, you can specify a host in the path name of the file database.

Wildcards

`osmv` can perform wildcard processing using regular expression wildcards `*`, `?`, `{}`, and `[]`.

UNIX

On UNIX, when operating on a rawfs database, you must enclose the wildcard in quotation marks (" ") or precede it with a backslash (\) to keep the shell from interpreting the wildcard as a shell wildcard.

API

See `os_dbutil::rename()` on page 189 in the *C++ API Reference*.

osprop

`osprop` is an ObjectStore/Single utility that performs a server checkpoint.

Note

If you are *not* using ObjectStore/Single, you must us the `ossvrchkpt` utility to perform a checkpoint; see `ossvrchkpt` on page 241.

Syntax

```
osprop [-f] server_log_name ...
```

Argument

`server_log_name`

Specifies the name of a server log file with unpropagated data.

Option

`-f`

Instructs `osprop` to ignore errors.

Description

The `osprop` utility ensures that committed data in the server logs is propagated to the affected databases. `osprop` is meaningful only when it is run as an ObjectStore/Single application.

`osprop` performs a function similar to `ossvrchkpt`. The difference is that `ossvrchkpt` propagates what it can, immediately. `osprop` propagates everything, guaranteed, and deletes the log when it is finished.

It is not usually necessary to run this utility because an ObjectStore/Single application that terminates normally conducts propagation and removes the log.

After `osprop` successfully propagates data in a log, it removes the log. Running `osprop` twice on the same server log is permissible.

API

Class: `objectstore`
Function: `propagate_log()`

Examples

The first example iteratively propagates committed data in the specified log files to the actual databases. Note that this is only meaningful if `osprop` is executed in an environment in which the ObjectStore/Single version of `libos` is used.

```
osprop log1 log2 ...
```

The next example propagates committed data in the specified log file if the file exists and is a valid log. Otherwise, the file is ignored.

```
osprop -f log3
```

osrecovr

The `osrecovr` utility copies (rolls forward) database modifications from archive log files created by `osarchiv` to the affected databases. See also Overview of the Backup/Restore Facility on page 48.

Syntax

```
osrecovr [options] [-f backup/log_file...]
[pathname_translation ...]
```

Argument

`pathname_translation ...`

Specifies a pair of path names. The first path name in the pair indicates the source of the database as recorded in the archive log or backup image. The second path name indicates the target; that is, the path name for the database after it is recovered.

You can specify zero, one, or more path name translations. Each path name can be a directory or a single database. However, you cannot specify a directory as the source and a database as the target.

If you do not specify at least one *`pathname_translation`*, all databases in the archive logs or backup images you specified are restored in their original locations.

Options

`-c`

Directs the `osrecovr` utility to apply each archive log snapshot and each backup image in its own transaction. The default is for all changes to be applied in a single transaction.

`-D` *`date`*

Specifies a date in the current locale — for example, *`mm/dd/yy`* in the United States. If the current locale is not known, executing the `osrecovr` utility with no options or arguments will display the usage message for the -D option with the correct date format. The `-D` option causes the `osrecovr` utility to roll forward all database changes committed before or on the specified date. The default is to roll forward to the last snapshot taken.

`-f` *`backup/log_file...`*

Specifies an archive log file or a backup image from which to recover committed database changes made since the last backup.

You can specify the `-f` option zero, one, or more times. The `osrecovr` utility processes the files in the order in which you specify them.

If you do not specify the `-f` option, you must specify the `-F` option.

You can mix specifications of `-f` and `-F`. The `osrecovr` utility processes them in the order in which you specify them.

Specifying a directory signals an error.

`-F recover_file`

Specifies the name of a file that contains a list of archive files or backup images from which to recover specified databases. If you specify a hyphen (`-`) as the recover file name, `osrecovr` reads from standard input.

The list contains one file path name per line. Leading and trailing white space is ignored.

If you specify the `-F` option, you can also specify `-f` with additional file names on the command line. You can mix specifications of `-f` and `-F`. `osrecovr` processes them in the order in which you specify them.

`-n`

Normally, if a directory is specified as the source of a recovery operation, all databases in the directory and its subdirectories are recovered. Including the `-n` option limits the recovery operation to databases contained in the named directory.

`-r time`

Specifies a recover-to time in the HH:MM:SS format. The `osrecovr` utility rolls forward all database changes committed before or at this time. The default is to roll forward to the last snapshot taken.

`-s size`

Optional. Specifies the size a chunk of data should be when restoring the target database. *`size`* is a number optionally appended with `k` or `m` to indicate kilobytes or megabytes. For example, `-s 1024k and -s 1m` both specify a maximum buffer size of 1 MB. If no unit is specified, `k` is presumed. The default value and minimum value is 512 KB. The maximum value is 1024 KB.

`-t`

Displays a list of databases contained in specified archive files.

`-T seconds`

Optional. Specifies the network timeout value. The value *`seconds`* is the amount of time. By default the value is the number of seconds, but `m`, `h`, or `d` can be appended to to the value to specify minutes, hours, or days, respectively. For example, `-T 60` and `-T 1m` both specify a timeout value of one minute. The default timeout value is 10 hours.

Description

The `osrecovr` utility can apply changes up to the time of the last snapshot in the archive log or to some earlier time that you specify.

The `osrecovr` utility can restore backups as well as recover data from archive logs, both in the same invocation.

When you run the `osrestore` or `osrecovr` utility, the operation is transaction protected. This means that if the operation fails, ObjectStore rolls databases back to the state they were in before the operation started.

ObjectStore applications cannot access databases that are being restored until the entire restoration process has finished.

Specify a *pathname_translation* when you want to restore

- One or more, but not all, databases in the backup image to their original locations
- One or more databases in the backup image to locations other than their original locations

Restoring from tape

When restoring data from tape, you must use the `osrestore` utility.

You must run the `osrestore` utility before the `osrecovr` utility if you performed both of the following steps:

1. You used the `osbackup` utility to back up a database.
2. You ran the `osarchiv` utility and used the same incremental backup record that you used for the `osbackup` utility.

Tradeoffs When Recovering in Several Transactions

You can specify the `-c` option to recover data in several transactions instead of one transaction. While this gives you flexibility, there is a tradeoff between the ability to roll back databases and the space needed in the log to record all modifications to databases being recovered.

For example, if you specify `-c` when you initiate `osrecovr`, ObjectStore recovers each snapshot in its own transaction. If the operation fails because of media failure while it is applying the last snapshot, ObjectStore rolls the databases back to the state they were in as of the last successfully applied snapshot.

However, suppose that each snapshot is 100 MB. This requires 100 MB of log space. If you ensure that the database does not exist when the recover operation starts, and if you apply all snapshots in a single transaction, all recovered data bypasses the log and goes directly to the database. Now if the operation fails, ObjectStore rolls all changes back, including the database creation.

The fundamental tradeoff is between the ability to roll back to a previous state and the resources needed to log the changes so that rollback is possible. In cases in which the size of the databases being recovered exceeds the size of the space available (or desirable) for logging, it is preferable

- To use a single transaction for the recovery operation
- Not to restore over existing databases

Examples

The following example lists archive contents:

```
% osrecovr -f /vancouver1/archives/1999011216.aaa -t
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
vancouver::/foo.db
vancouver::/dbdir/bar.db
vancouver::/dbdir/foo.db
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
```

```
%
```

This next example recovers from a single archive:

```
% osrecovr -f /vancouver1/archives/1999011216.aaa
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
Target time: Tue Jan 12 17:28:22 1999
Recovered to time Tue Jan 12 16:25:27 1999
Recovered to time Tue Jan 12 16:25:57 1999
Recovered to time Tue Jan 12 16:26:11 1999
Restoring 452 sectors to database
"vancouver:/vancouver1/dbdir/foo.db"
Recovered to time Tue Jan 12 16:26:41 1999
Recovered to time Tue Jan 12 16:27:13 1999
Recovered to time Tue Jan 12 16:27:43 1999
Recovered to time Tue Jan 12 16:28:14 1999
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
%
```

The next example illustrates how to recover back to a specific place. It restores all databases to their location and state as of 16:25:27 on January 12, 1999.

```
% osrecovr -f /vancouver1/archives/1999011216.aaa -r 16:25:27
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
Target time: Tue Jan 12 16:25:27 1999
Recovered to time Tue Jan 12 16:25:27 1999
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
```

Recovering from multiple archive files:

```
% cat ./archive_list
/vancouver1/archives/1999011216.aaa
/vancouver1/archives/1999011216.aab
/vancouver1/archives/1999011216.aac
```

```
% osrecovr -t -F ./archive_list
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
vancouver::/foo.db
vancouver::/dbdir/bar.db
vancouver::/dbdir/foo.db
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
```

```
% osrecovr -F ./archive_list
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
Target time: Tue Jan 12 17:27:01 1999
Recovered to time Tue Jan 12 16:25:27 1999
Recovered to time Tue Jan 12 16:25:57 1999
Recovered to time Tue Jan 12 16:26:11 1999
Restoring 452 sectors to database
"vancouver:/vancouver1/dbdir/foo.db"
Recovered to time Tue Jan 12 16:26:41 1999
Recovered to time Tue Jan 12 16:27:13 1999
Recovered to time Tue Jan 12 16:27:43 1999
Recovered to time Tue Jan 12 16:28:14 1999
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
Auto switching to volume #2 (/vancouver1/archives/1999011216.aab).

Recovering from volume #2 (/vancouver1/archives/1999011216.aab)...
Recovered to time Tue Jan 12 16:28:21 1999
Recovered to time Tue Jan 12 16:28:35 1999
Recovered to time Tue Jan 12 16:28:37 1999
Recovered to time Tue Jan 12 16:28:38 1999
Recovered to time Tue Jan 12 16:28:40 1999
Recovered to time Tue Jan 12 16:28:41 1999
```

```
Recovered to time Tue Jan 12 16:28:49 1999
Recovered to time Tue Jan 12 16:28:55 1999
Recovered to time Tue Jan 12 16:29:01 1999
Recovered to time Tue Jan 12 16:29:06 1999
Recovered to time Tue Jan 12 16:29:12 1999
Recovered to time Tue Jan 12 16:29:17 1999
Recovered to time Tue Jan 12 16:29:23 1999
Recovered to time Tue Jan 12 16:29:28 1999
Recovered to time Tue Jan 12 16:29:34 1999
Recovered to time Tue Jan 12 16:29:39 1999
Recovered to time Tue Jan 12 16:29:43 1999
Recovered to time Tue Jan 12 16:29:44 1999
Recovered to time Tue Jan 12 16:29:49 1999
Recovered to time Tue Jan 12 16:29:55 1999
Recovered to time Tue Jan 12 16:30:01 1999
Closing volume #2 (/vancouver1/archives/1999011216.aab).
Auto switching to volume #3 (/vancouver1/archives/1999011216.aac).

Recovering from volume #3 (/vancouver1/archives/1999011216.aac)...
Recovered to time Tue Jan 12 16:31:04 1999
Recovered to time Tue Jan 12 16:31:06 1999
Closing volume #3 (/vancouver1/archives/1999011216.aac).
%
```

Recovering to a date and time:

```
% osrecovr -F ./archive_list -D 1/30/99 -r 16:27:43
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
Target time: Tue Jan 12 16:27:43 1999
Recovered to time Tue Jan 12 16:25:27 1999
Recovered to time Tue Jan 12 16:25:57 1999
Recovered to time Tue Jan 12 16:26:11 1999
Restoring 452 sectors to database
"vancouver:/vancouver1/dbdir/foo.db"
Recovered to time Tue Jan 12 16:26:41 1999
Recovered to time Tue Jan 12 16:27:13 1999
Recovered to time Tue Jan 12 16:27:43 1999
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
%
```

Recovering to a time today:

```
% osrecovr -F ./archive_list -r 16:27:43
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
Target time: Tue Jan 12 16:27:43 1999
Recovered to time Tue Jan 12 16:25:27 1999
Recovered to time Tue Jan 12 16:25:57 1999
Recovered to time Tue Jan 12 16:26:11 1999
Restoring 452 sectors to database
"vancouver:/vancouver1/dbdir/foo.db"
Recovered to time Tue Jan 12 16:26:41 1999
Recovered to time Tue Jan 12 16:27:13 1999
Recovered to time Tue Jan 12 16:27:43 1999
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
%
```

The next example illustrates recovering a single database. It makes `vancouver::/bar.db` equal to `vancouver::/foo.db` as of 16:27:43 today.

```
% osrecovr -F ./archive_list -r 16:27:43 vancouver::/foo.db \
vancouver::/bar.db
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
Target time: Tue Jan 12 16:27:43 1999
```

```
Recovered to time Tue Jan 12 16:25:27 1999
Recovered to time Tue Jan 12 16:25:57 1999
Recovered to time Tue Jan 12 16:26:11 1999
Recovered to time Tue Jan 12 16:26:41 1999
Recovered to time Tue Jan 12 16:27:13 1999
Recovered to time Tue Jan 12 16:27:43 1999
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
```

```
% osls vancouver::/
bar.db
dbdir/
foo.db
%
```

```
% cat ./archive_list
/vancouver1/archives/1999011216.aaa
/vancouver1/archives/1999011216.aab
/vancouver1/archives/1999011216.aac
```

```
% osrecovr -t -F ./archive_list
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
vancouver::/foo.db
vancouver::/dbdir/bar.db
vancouver::/dbdir/foo.db
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
%
```

Examples of recovery failures

Nonexistent database:

```
% osrecovr -f /vancouver1/archives/1999011216.aaa -r 16:25:27 \
vancouver::/asdla.db vancouver::/as.db
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
Recover failed: Database vancouver::/asdla.db does not exist in this
backup image
%
```

Day not in the archives:

```
% osrecovr -F ./archive_list -D 1/30/99
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
Target time: Mon Jan 11 17:29:08 1999
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
%
```

Day/year not in archives:

```
% osrecovr -F ./archive_list -D 1/30/98
Recovering from volume #1 (/vancouver1/archives/1999011216.aaa)...
Target time: Sun Jan 11 17:29:51 1998
Closing volume #1 (/vancouver1/archives/1999011216.aaa).
%
```

osreplic

The `osreplic` utility replicates and maintains multiple copies of a database.

Syntax

```
osreplic [options] -a archive_record_file src_path1 dest_path1
[src_path2 dest_path2 ...]
```

Arguments

`src_path1`

Specifies the directory path or database name of the database or databases you want to replicate.

`dest_path1`

Specifies the directory path or database name of the replica database or databases.

Options

`-a` *`archive_record_file`*

Required. Specifies the archive record file. If the file does not exist, it is created.

`-B` *`size`*

Optional. Controls the amount of transient workspace available to the source server.

`size` is a number optionally appended with `k` or `m` to indicate kilobytes or megabytes, respectively. If no unit is specified, `m` is presumed. For example, `-B 1024k`, `-B 1m`, and `-B 1` each specify a maximum buffer size of 1 MB. The default value is 1 MB and the maximum value is 2 MB.

`-c`

Optional. Provides a clean-up handler if you use Control-C to exit the utility.

`-C`

Optional. Enables the interactive command-loop feature. This feature is disabled by default. See Commands on page 206 for the commands available in interactive mode.

`-i` *`interval`*

Optional. Sets the interval between snapshots. The default is 600 seconds. A copy is made immediately after `osreplic` is initiated, then every *`interval`* thereafter.

Intervals are specified with integer values in seconds. For example, `-i 60` specifies an interval of 60 seconds.

`-I` *`input_file`*

Optional. Source and destination databases and directories can be specified with a separate input file as well as on the command line. Each line in the input file should consist of a source path followed by a target path. If a directory is specified, its contents are added to the source set.

`-o` *`output_file`*

Optional. Redirects output to *`output_file`*.

`-p`

Optional. Sets permissions on the replica to match those of the master (rawfs only).

`-P`

Optional. Specifies that `osreplic` will use data compression.

`-r`

Optional. Enables recursive processing of rawfs directories.

`-R` *`size`*

Optional. Specifies the size a chunk of data should be when copying from the source database. *`size`* is a number optionally appended with `k` or `m` to indicate kilobytes or megabytes. For example, `-R 1024k and -R 1m` both specify a maximum buffer size of 1 MB. If no unit is specified, `k` is presumed. The default value and minimum value is 256 KB. The maximum value is 102400 KB.

`-s` *`size`*

Optional. Specifies the size a chunk of data should be when updating the target database. *`size`* is a number optionally appended with `k` or `m` to indicate kilobytes or megabytes. For example, `-s 1024k and -s 1m` both specify a maximum buffer size of 1 MB. If no unit is specified, `k` is presumed. The default value and minimum value is 512 KB. The maximum value is 1024 KB.

`-T` *`seconds`*

Optional. Specifies the network timeout value. The value *`seconds`* is the amount of time. By default the value is the number of seconds, but `m`, `h`, or `d` can be appended to to the value to specify minutes, hours, or days, respectively. For example, `-T 60` and `-T 1m` both specify a timeout value of one minute. The default timeout value is 10 hours.

`-v`

Optional. Enables verbose output.

`-x`

Optional. Prohibits clients from using the replica until `osreplic` terminates.

Commands

You can execute the following commands when you use `osreplic` in interactive mode. The utility processes the commands between snapshots.

`i` *`interval`*

Interval — Changes the interval between snapshots. Specify an integer for *`interval`*. You can append the letter `m`, `h`, or `d` to indicate minutes, hours, or days. For example, `i 60` and `i 1m` both specify an interval of one minute. When *`interval`* is `0`, `osreplic` takes a single snapshot and terminates.

You can specify `i` without an integer to display the current interval.

`s`

Statistics— Display replication statistics..

`q`

Quit — Terminate `osreplic` after completing the current replication transaction.

Description

The ObjectStore replicator produces a continuously updated copy (or replica) of one or more user databases. The utility works by coordinating the actions of a source ObjectStore server running an archive logger and of a target ObjectStore server. It provides a read-only (MVCC) copy of a database that is updated dynamically from the master database. ObjectStore rawfs databases and rawfs directories can be replicated, as well as ObjectStore file databases. Native file system directories cannot be replicated.

When you start the replicator, you specify a set of sources and destinations for replicated databases on the command line or with a separate input file by using the `-I` option. The master/replica pair is specified by path name with the *`src_path`* and *`dest_path`* arguments, including the target host, if needed. The list of databases cannot be changed after the replicator is started.

`src_path` and *`dest_path`* can be a directory path or a database name. You can also use UNC path names (on Windows platforms), server relative path names, or local path names. However, you cannot use UNC path names as destinations.

You must also specify an *`archive_record_file`* with the `-a` option. The `osreplic` utility uses this information to determine those clusters and segments within a database that have been modified since the last replication. This file is identical to the archive record file for `osarchiv`. At specified intervals, the replicator takes a snapshot of the databases and sends the changed data to the target host so that the data is applied to the replica. All committed user data is replicated.

When a transaction commits, its modifications are not replicated until the next snapshot. As with the `osarchiv` utility, the `-i` option can be used to adjust the trade-off between keeping the replica more up to date and minimizing the volume of data transmitted. For more information, see Tradeoffs for Obtaining the Results You Need on page 141.

The `-R` and `-s` options can be used to improve performance over wide area networks with wide bandwidth but long latency. Increasing the size of the chunks of data will reduce the number of chunks that need to sent across the network.

Owner, group, and mode permissions can also be copied for rawfs databases. However, neither rawfs directory permissions nor segment-level permissions are copied. Additionally, no file database permissions are copied. Operations such as `osrm` are not propagated to the replica. If a failover occurs while `osreplic` is updating a database, `osreplic` will use the archive record file and continue replicating the data from where the last snapshot left off.

API

See `os_replicator` on page 315 in the *C++ API Reference*.

osrestore

The `osrestore` utility copies databases from backup storage locations to your disk or rawfs. Backups must have been created with the `osbackup` utility. See also Overview of the Backup/Restore Facility on page 48.

Syntax

```
osrestore [options] -f backup_file [-f backup_file]... [pathname_
translation]...
```

Argument *pathname_translation...*

Specifies a pair of path names separated by a space. The first path name in the pair indicates the source of the database as recorded in the backup image. The second path name indicates the target; that is, the path name for the database after it is restored.

You can specify zero, one, or more path name translations. Each path name can be a directory or a single database. However, you cannot specify a directory as the source and a database as the target.

If you do not specify at least one *pathname_translation*, all databases in the backup image are restored in their original locations.

Options `-a`

Optional. Aborts the restore operation if the utility cannot open the restore device. This raises an exception that indicates the problem.

By default, if `osrestore` fails to open the device, it displays a message and waits for you to correct the problem.

The utility will fail to open the restore device if, for example, no tape is loaded.

`-b` *blocking_factor*

Optional. Specifies a blocking factor to use for tape input and output. This option applies only when you are restoring data from a tape. The blocking factor is in units of 512-byte blocks. The default on UNIX is 126 blocks. The maximum blocking factor is 512 blocks.

`-c`

Optional. Provides a clean-up handler if you use Control-C to exit the utility.

`-f` *backup_file*

Required. Specifies a file or tape device that contains a backup image from which to restore databases. You can specify the `-f` option one or more times.

On UNIX systems, you can specify `-f -` (hyphen) to indicate `stdin`.

`-I` *input_file*

Optional. In addition to specifying the database translation pair on the command line, you can use this option to specify the paths of the source and target files in *input_file*, a separate input file. Each line in the input file must consist of one

path name. The first line should indicate the source of the database as recorded in the backup image. The next line should indicate the target path — that is, the path name for the database after it is restored.

The `-I` option provides an alternative to listing database translation pairs on the command line and is useful when you want to list more pairs than can be accomodated on the command line . For more information, see the description of the *`pathname_translation`* argument to `osrestore`.

`-n`

Optional. Normally, if a directory is specified as a source for `osrestore`, all databases in the directory and its subdirectories are restored. Including the `-n` option limits the operation to databases in the named directory.

`-0` (zero)

Optional. Restores the database image specified with the `-f` flag, then exits. There is no prompt for additional volumes.

`-o` *`output_file`*

Optional. Redirects output to *`output_file`*.

`-p`

Optional. The `-p` (permissions) option causes `osrestore` to restore database permissions for the rawfs stored in the archive log file for the database being restored.

`-s` *`size`*

Optional. Specifies the size a chunk of data should be when restoring the target database. *`size`* is a number optionally appended with `k` or `m` to indicate kilobytes or megabytes. For example, `-s 1024k and -s 1m` both specify a maximum buffer size of 1 MB. If no unit is specified, `k` is presumed. The default value and minimum value is 512 KB. The maximum value is 1024 KB.

`-S` *`exec_command_name`*

Optional. Specifies the path of a command to be executed when the `osrestore` utility reaches the end of the media. This command should mount the next volume before returning. The exit status from this command must be `0` or the restore operation aborts.

`-t`

Optional. Displays a list of databases in the backup image.

`-T` *`seconds`*

Optional. Specifies the network timeout value. The value *`seconds`* is the amount of time. By default the value is the number of seconds, but `m`, `h`, or `d` can be appended to to the value to specify minutes, hours, or days, respectively. For example, `-T 60` and `-T 1m` both specify a timeout value of one minute. The default timeout value is 10 hours.

Description

ObjectStore applications cannot access databases that are being restored until the entire restoration process has finished.

Specify a *pathname_translation* when you want to restore

- One or more, but not all, databases in the backup image to their original locations
- One or more databases in the backup image to locations other than their original locations

Restore procedure

To restore databases, begin with a level 0 backup image. The `osrestore` utility prompts for incremental backup images you might want to apply after this. Not all incremental backups need to be applied. To determine which incremental backups to apply, list the backup levels in chronological order, starting with the level 0 backup. For example, suppose you performed the following backups:

- Level 0 backup on Monday
- Level 5 on Tuesday
- Level 6 on Wednesday
- Level 2 on Thursday
- Level 4 on Friday

Your list would look like this: 0, 5, 6, 2, 4.

Scanning the list from right to left, find the lowest incremental backup level greater than 0, in this case, the level 2 backup made on Thursday. To restore databases to their state as of the backup on Friday, apply the level 0 backup and the incremental backups made at levels 2 and 4, in that order.

Block size

The block size must be 512 bytes or less. The `osrestore` utility cannot work when the block size is greater than 512 bytes.

Comparing databases

You might want to have two copies of the same database for verification purposes: a restored version and the original version. Following is a sample command line for doing this. In this example, `backup.img` contains `foo::/db`. The path name translation does the job in one step.

```
osrestore -f backup.img foo::/db foo::/restore.db
```

Windows to UNIX path name translation example

You must specify a path name translation when you restore or recover data on an architecture that is different from the architecture on which you are restoring the data. An example is a Windows to UNIX path name translation. The backup image being restored is `/tmp/my.img`. The interaction is on a UNIX system. You need not do anything special when you make the backup on the Windows system.

In the first interaction, the command line specifies the `-t` option, which instructs the `osrestore` utility to list the databases in the specified backup image. Nothing is actually restored. The only database in the backup image is `mckinley:e:\r4tsd_data\arch.0`. This is a Windows database, and the following example shows that the `osrestore` utility on a UNIX system translates it to `mckinley:e:/r4tsd_`

data/arch.0. The utility automatically translates backslashes (\) to forward slashes (/).

```
% osrestore -f /tmp/my.img -t
Recovering from volume #1 (/tmp/my.img)...
mckinley:e:/r4tsd_data/arch.0
Closing volume #1 (/tmp/my.img).
%
```

In the second interaction, the command line specifies the path name translation mckinley:e:/r4tsd_data/ /recovery. This instructs the osrestore utility to copy all files in the backup image in the mckinley:e:/r4tsd_data/ directory to the /recovery directory on the local machine. In this example, this is only arch.0.

```
% osrestore -f /tmp/my.img mckinley:e:/r4tsd_data/ /recovery
Recovering from volume #1 (/tmp/my.img)...
Restoring 3175 sectors to database "vancouver:/recovery/arch.0"
Recovered to time Fri Mar 3 14:07:24 1999

Do you wish to restore from any additional incremental backups?
(yes/no):
no
Closing volume #1 (/tmp/my.img).
%
```

Examples

The following examples illustrate some uses of osrestore. Although it is not shown, osrestore prompts you to indicate whether you want to restore from incremental backups. The examples are UNIX examples; however, they would be the same on any platform, except for the file name format.

The following example displays a list of databases in the backup.img backup image:

```
% osrestore -t -f /backup.img
::eudyp:/test/
::eudyp:/test: data1.odb data2.odb data3.odb
::cleopat:/results/
::cleopat:/results: r1.odb r2.odb r3.odb
```

The following example indicates that the backup image contains six file databases. Three are in the /test directory; they were backed up on host eudyp. Three are in the /results directory; they were backed up on host cleopat.

This example shows how to copy backups to new servers. It restores all databases on server eudyp to server kellen, and all databases on server cleopat to server eudyp:

```
% osrestore -f backup.img   eudyp:/ kellen:/   cleopat:/ eudyp:/
restoring "::eudyp:/test/data1.odb" to "::kellen:/test/data1.odb"
restoring "::eudyp:/test/data2.odb" to "::kellen:/test/data2.odb"
restoring "::eudyp:/test/data3.odb" to "::kellen:/test/data3.odb"
restoring "::cleopat:/results/r1.odb" to "::eudyp:/results/r1.odb"
restoring "::cleopat:/results/r2.odb" to "::eudyp:/results/r2.odb"
restoring "::cleopat:/results/r3.odb" to "::eudyp:/results/r3.odb"
```

The next example shows how to change servers and directories. It restores all databases in the /test directory on server eudyp into the /test-copy directory on server kellen:

```
% osrestore -f backup.img  eudyp:/test  kellen:/test-copy
```

```
restoring "::eudyp:/test/data1.odb" to
"::kellen:/test-copy/data1.odb"
restoring "::eudyp:/test/data2.odb" to
"::kellen:/test-copy/data2.odb"
restoring "::eudyp:/test/data3.odb" to
"::kellen:/test-copy/data3.odb"
```

The next example shows how to restore a single database. It restores the database `eudyp:/test/data1.odb` to `/tmp`:

```
% osrestore -f backup.img   eudyp:/test/data1.odb  eudyp:/tmp
restoring "::eudyp:/test/data1.odb" to "::eudyp:/tmp/data1.odb"
```

The next example illustrates restoring to a source, with one exception. It restores everything in the `/test` directory on server `eudyp` to its original location, except `data1.odb`. This database gets restored in the `/example` directory on server `cleopat`. In this example, the order of the path name translations is important. Specify specific path names before you specify directories that include those path names.

```
% osrestore -f backup.img eudyp:/test/data1.odb \
cleopat: /example  eudyp:/test  eudyp:/test
restoring "::eudyp:/test/data1.odb" to "::cleopat:/example/data1.odb"
restoring "::eudyp:/test/data2.odb" to "::eudyp:/test/data2.odb"
restoring "::eudyp:/test/data3.odb" to "::eudyp:/test/data3.odb"
```

The following example illustrates restoring to the source. It restores the entire backup image to its original location.

```
% osrestore -f backup.img
restoring "::eudyp:/test/data1.odb" to "::eudyp:/test/data1.odb"
restoring "::eudyp:/test/data2.odb" to "::eudyp:/test/data2.odb"
restoring "::eudyp:/test/data3.odb" to "::eudyp:/test/data3.odb"
restoring "::cleopat:/results/r1.odb" to "::cleopat:/results/r1.odb"
restoring "::cleopat:/results/r2.odb" to "::cleopat:/results/r2.odb"
restoring "::cleopat:/results/r3.odb" to "::cleopat:/results/r3.odb"
```

The following illustrates restoring everything to a local directory. It restores the entire backup image into the `/examples` directory on the local host (`twinkie`).

```
% osrestore -f back.img eudyp:/test \ /examples  cleopat:/results
/examples
restoring "::eudyp:/test/data1.odb" to
"::twinkie:/examples/data1.odb"
restoring "::eudyp:/test/data2.odb" to
"::twinkie:/examples/data2.odb"
restoring "::eudyp:/test/data3.odb" to
"::twinkie:/examples/data3.odb"
restoring "::cleopat:/results/r1.odb" to "::twinkie:/examples/r1.odb"
restoring "::cleopat:/results/r2.odb" to "::twinkie:/examples/r2.odb"
restoring "::cleopat:/results/r3.odb" to "::twinkie:/examples/r3.odb"
```

osrm

The `osrm` utility removes databases and rawfs links from servers.

Syntax

`osrm` *`options pathname...`*

Note that you cannot concatenate options with the `osrm` utility — that is, `osrm -r -f` is acceptable, but `osrm -rf` is not.

Argument

`pathname...`

Specifies the file or rawfs database or directory, or rawfs link, that you want to remove. You can specify one or more, and you can specify both file and rawfs databases and directories and rawfs links in the same operation. You must specify the `-r` option if you want to remove a rawfs directory.

Options

`-f`

Forces execution of the utility and does not display an error message if the specified database is not found or cannot be removed. This option is required when you want to remove nondatabase files from the native file system.

`-i`

Specifies interactive mode. ObjectStore prompts you to confirm for each specified database that you really want to remove it.

`-r`

Recursively removes all databases in the specified directory.

Description

To remove a database, you must have write permission to its directory, but you need not have write access to the database itself.

If you specify more than one database to be removed and for some reason ObjectStore cannot remove at least one of the databases, ObjectStore does not remove any of the databases.

If a database is open when you remove it with the `osrm` utility, ObjectStore does not actually remove it until it is closed. Transactions can update the removed database until the database is closed.

For file databases, the `osrm` utility calls the native remove utility.

Wildcards

The `osrm` utility can perform wildcard processing using regular expression wildcards `*`, `?`, `{}`, and `[]`.

UNIX

On UNIX systems, when you are operating on a rawfs database, you must enclose the wildcard in quotation marks (" ") or precede it with a backslash (\) to keep the shell from interpreting the wildcard as a shell wildcard.

API

See `os_dbutil::remove()` on page 189 in the *C++ API Reference*.

osrmdir

The `osrmdir` utility removes a directory from the rawfs.

Syntax

```
osrmdir directory
```

Argument

directory

Specifies the path of the directory that you want to remove from the rawfs.

Description

If you want to remove a directory from the rawfs, the directory must be empty. Also, you must have write permission for the parent directory. You do not need write permission for the directory you are removing.

You also can use `osrmdir` to remove a local directory. When you specify a local directory, you cannot specify a remote file-server host in the path name of the local directory. The `osrmdir` utility passes the operation to a local native utility. If you specify a remote file-server host name, ObjectStore informs you that you specified an illegal path name.

Wildcards

The `osrmdir` utility can perform wildcard processing using regular expression wildcards `*`, `?`, `{}`, and `[]`.

UNIX

On UNIX, you must enclose the wildcard with quotation marks (" ") or precede it with a backslash (\) to keep the shell from misinterpreting the asterisk as a shell wildcard.

API

See `os_dbutil::rmdir()` on page 189 in the *C++ API Reference*.

osscheq

The `osscheq` utility compares two schemas.

Syntax

```
osscheq [-quiet] db1 db2
```

Arguments

`db1 db2`

Specifies the paths of the two databases you want to compare. Each database can contain an application schema, a compilation schema, or a database schema. If the database contains a database schema, it can be local or remote.

Options

`-layout_only`

Specifies that you want to compare the databases only on the basis of the layouts of their persistent objects.

`-quiet`

Returns a value of `0` if the databases are compatible and a nonzero value if the databases are not compatible. When you do not specify this option, the utility displays messages explaining why the schemas are different. There is no other output.

Description

The `osscheq` utility is useful when you suspect that a change to a schema causes it to be incompatible with the other schemas in an application. It is best to detect an incompatibility as early as possible. When schemas are not compatible, execution of the application fails because of a schema validation error.

Comparison technique

The comparison technique depends on the types of schemas being compared. When `osscheq` compares compilation, library, application, or component schemas, ObjectStore uses the technique used by the schema generator to build the schemas. When one of the schemas being compared by `osscheq` is a database schema, the comparison technique is the same as that used to validate an application schema when the application accesses a database.

Examples

Suppose the database `test1` contains the following definition for C:

```
class C {
  int del ;
  int mod ;
} ;
```

Database `test2` defines C as follows:

```
class C {
  int add ;
  char* mod ;
} ;
```

Invoke the `osscheq` utility as follows:

```
osscheq test1 test2
```

The result is the following output:

```
The following class definitions in test1 and test2 were inconsistent:
C ( C.del was deleted, the type of C.mod changed (from int to char*),
and C.add was added)
```

Schema checking done by the schema generator is a stricter form of checking than that used to validate an application against a database. The latter form of checking is the minimal checking required to ensure that the application and the database use the same layout for all shared classes.

osserver

The `osserver` utility starts the server. Starting the server varies from platform to platform. See Chapter 7, Managing ObjectStore on UNIX, on page 299, or Chapter 8, Managing ObjectStore on Windows, on page 317, for information about starting the server on the respective platforms.

Note

Server error messages are routed to `%OS_TMPDIR%\OSSERVER.TXT` (Windows) or `/tmp/oss_out` (UNIX); for more information, see "Finding Files Containing ObjectStore Messages" in Chapter 7, Managing ObjectStore on UNIX, on page 299, or Chapter 8, Managing ObjectStore on Windows, on page 317.

Syntax

```
osserver options
```

Options

Ordinarily, you use server parameters to control the way the server functions. However, you can also specify options when you execute the `osserver` utility.

`-c`

Performs a checkpoint by forcing all data to be propagated from the log to the database. The server does not start after this checkpoint.

`-con` or
`-console`

(Windows only) Runs the server as a console-mode application. You cannot use this option as a Start Parameter for a Service.

`-d` *`int`*

Starts the server in debug mode. Specify an integer from 1 through 9. The larger the number, the more information ObjectStore provides. You also can specify the `-F` or `-con` option so that ObjectStore displays the information on the screen.

ObjectStore copies debug output to the standard server output file as well as standard output.

`-F`

(All platforms except Windows) Runs the server as a foreground process. ObjectStore displays the information on the screen. This reverses the normal behavior, in which the server runs as a background process.

When it is run without `-F`, `osserver` returns `0` from a successful startup of the background daemon; otherwise it returns `1`.

`-hostname` *`logical_hostname`*

Specifies a logical host name. If the server is started with the `-hostname` option on the command line and *`logical_hostname`* does not specify the local host, the server does not listen to local network ports, only IP ports. The logical host name must be defined in the appropriate network directory, such as DNS.

If you are using a cluster operating system, each `osserver` process must be started with the `-hostname` option so that each `osserver` process knows its logical host

name. When the cluster operating system moves the osserver process to another cluster node, it also moves the logical host name to the new node; see Cluster-Managed Failover on page 288.

`-i`

If you have a rawfs, this option initializes the rawfs and the server log file if it is in the rawfs, with a confirmation prompt. In the absence of a rawfs, this option initializes the server log file, with a confirmation prompt. Note that the server does not start after this initialization.

Caution

Use this option carefully. Initializing a log file that contains unpropagated data will prevent that information from being propagated to the databases and can result in a corrupted and unusable database. Using `-i` to initialize the rawfs will also delete any databases in the rawfs.

`-I`

This option is the same as the `-i` option, but without the confirmation prompt. That is, if you have a rawfs, this option initializes the rawfs and the server log file if it is in the rawfs. In the absence of a rawfs, this option initializes the server log file. Note that the server does not start after this initialization.

Caution

Use this option carefully. Initializing a log file that contains unpropagated data will prevent that information from being propagated to the databases and can result in a corrupted and unusable database. Using `-I` to initialize the rawfs will also delete any databases in the rawfs.

`-M`

"Mend" the rawfs file system by deleting any databases that are so badly corrupted that they would prevent the server from starting up.

Caution

Use this option carefully. Contact Technical Support if you think you need to use it.

`-p` *`pathname`*

(UNIX only.) Specifies a file containing server parameter settings that override the default settings. If you do not specify this option, ObjectStore uses the default parameter file `$OS_ROOTDIR/etc/`*`host`*`_server_parameters`, where *`host`* is the name specified with the `-hostname` or `-server_name` option, or is the name of the host machine on which this `osserver` process is running.

`-r` *`registry_location`*

(Windows only.) Specifies the Windows registry location relative to `HKEY_LOCAL_MACHINE\SOFTWARE` where the server should look for server parameters. If the `-r` option were followed by the string `my_location`, the server would look in the following location:

```
HKEY_LOCAL_MACHINE\SOFTWARE\my_location\Server
```

By default, the registry location for ObjectStore is:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore 6.0
```

For information about using the `-r` option, see Changing the Registry Location for ObjectStore (Windows Only) on page 325.

`-ReallocateLog`

Reallocates (that is, deletes and recreates) the server transaction log file. The server does not start after this reallocation.

Caution

Use this option carefully. Reallocating a log file that contains unpropagated data will prevent that information from being propagated to the databases and can result in a corrupted and unusable database. Call Technical Support for assistance.

`-server_name` *`logical_server_name`*

Specifies a logical server name. The `-server_name` option tells the `osserver` process its logical server name. Use this option when running four osserver processes with ObjectStore-managed failover; see Four osserver Processes, Two Logical Servers on page 287. The logical server name must be defined for clients in a locator file, as described in SERVER Declaration (ObjectStore-Managed Failover Only) on page 294.

`-upgradeRAWFS`

Use this option when enabling or disabling ObjectStore-managed failover. When enabling failover, run `osserver` with this option to add a heartbeat either to the heartbeat file or to a rawfs partition. If you want to reconfigure ObjectStore without failover and the heartbeat is stored in a rawfs partition, you must run `osserver` with this option to remove the heartbeat.

For more information, see Restarting the Server (ObjectStore-Managed Failover Only) on page 290 and Removing Failover (ObjectStore-Managed Failover Only) on page 293.

`-v`

Displays server parameter values at startup.

Description

The `osserver` utility starts an ObjectStore server. After starting, the server displays the message `Server started` on the system (host) console to let you know that it is ready to accept requests from clients on the network.

Initializing the Transaction Log

When using `-i`, `-I`, or `-ReallocateLog` to initialize the transaction log, be sure to move all data out of the log before initializing it; see ossvrchkpt on page 241.

When you specify `-i`, ObjectStore responds with the following:

```
You have asked for initialization which will create a new transaction
log, deleting any old log that might exist, thus destroying any
recovery data in the old log. This might leave some file databases in
a broken state. Are you sure that you want to create a new log?
```

When you specify `-I`, this message does not appear.

When you initialize the transaction log, anything in the log is lost. The server resizes the log to the values specified by the `Log Data Segment Initial Size` and `Log Record Segment Initial Size` server parameters. This action frees space for use by other files.

Initializing the rawfs

If you have a rawfs, specifying the `-i` or `-I` option with `osserver` initializes the rawfs and also the transaction log if it is in the rawfs. Be sure to back up all data in the rawfs before initializing it; see osbackup on page 143.

When you specify `-i`, ObjectStore responds with the following:

```
The entire ObjectStore file system on this host is about to be
initialized, thus destroying any data currently in it. Unless you have
a backup of the ObjectStore file system, it will be impossible to
recover the old contents once the initialization is started. Are you
sure that you want to reinitialize the ObjectStore file system?
```

When you specify `-I`, this message does not appear.

Note that if you run `osserver` with `-i` or `-I` and you do not have a rawfs, `osserver` initializes only the transaction log.

Debug Mode

Running the server with the `-d` option (debug mode) will impact the server's performance but has no other effect on clients. You should run the server in debug mode only when you are experiencing a problem, especially when you plan to contact Technical Support about a server problem. In most cases, specify `-d 9` so you can obtain all possible debug information.

When you no longer need the debug output, shut down the server and restart it without the debug option or run `ossvrdebug 0` to turn off debug mode. For more information, see ossvrdebug on page 244.

Platform Information

For platform-specific information about the `osserver` utility, see the following:

- UNIX: Starting the Server on page 302
- Windows: Starting the Server on page 319

ossetasp

The `ossetasp` utility patches an executable so that it looks for its application schema in a database that you specify.

Syntax

```
ossetasp {-p executable_file | path_of_executable db_path}
```

Arguments

`path_of_executable`

Specifies the path of an executable. On Windows systems, this can also be the path name of a DLL. On UNIX systems, *`path_of_executable`* can be the path name of a shared library.

`db_path`

Specifies the path of an application schema database. ObjectStore patches the specified executable so it uses this application schema.

Option

`-p` *`executable_file`*

Instructs `ossetasp` to display the path name of the specified executable's application schema database.

Description

When the schema generator generates an application schema, ObjectStore stores the actual string given as the `-asdb` argument to `ossg` (or the `-final_asdb` argument, if specified). When the application starts, it uses that string to find the application schema database.

When you move or copy an ObjectStore application to a machine that is not running a server, leave the application schema database on the server host. Normally, the application schema database must be local to the server.

After you copy or move an application to another machine, you must patch the executable to enable it to find the application schema database. Run the `ossetasp` utility with the absolute path name of the application schema database. Be sure to specify the name of the server host.

Instead of patching the executable, you can use one of the following environment variables:

- `OS_SCHEMA_PATH`
- The platform-dependent variables `PATH`, `LD_LIBRARY_PATH`, `LD_LIBRARYN32_PATH`, `LIBPATH`, or `SHLIB_PATH`
- `OS_LIBDIR`

A locator file allows a database and its application schema to be on a machine other than the server host. See Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 265.

Windows

On Windows systems, you can perform the `ossetasp` utility on any EXE or DLL that contains a schema (that is, that has a schema object file produced by `ossg` linked into it).

API

See `objectstore::get_application_schema_pathname()` on page 64 and `objectstore::set_application_schema_pathname()` on page 83 in the *C++ API Reference*.

ossetrsp

The `ossetrsp` utility specifies a new path name for the schema associated with a particular database.

Syntax

```
ossetrsp {-p | schema_db_path} db_path
```

Arguments

`schema_db_path`

Specifies a new path name for the schema database used by the database specified by *`db_path`*.

`db_path`

Specifies the path name of a database whose schema database path name you want to either change or display.

Option

`-p` *`db_path`*

Displays the path name of the schema database used by the database specified by *`db_path`*, if *`db_path`* is a database that stores its schema remotely. If it is not, ObjectStore displays a message informing you that *`db_path`* is not a database whose schema is stored in another database.

Description

A database can store its schema in a separate schema database. The schema database contains all schema and relocation metadata. The main database contains everything else.

When needed

If you move the schema database, you must execute `ossetrsp` or use `os_database::set_schema_database()` to inform ObjectStore of the schema database's new path name.

If you copy the schema database with an operating system command or an ObjectStore utility, you can execute `ossetrsp` or use `os_database::set_schema_database()` to inform ObjectStore of the schema database's new path name.

You cannot associate an entirely new schema database with the main database. You can only change the path name of the original schema database by moving or copying the original schema database.

API

See `os_database::get_schema_database()` on page 160 and `os_database::set_schema_database()` on page 169 in the *C++ API Reference*.

ossevol

The `ossevol` utility modifies a database and its schema so that it matches a revised application schema. It handles many common cases of schema evolution. For more complicated evolutions, including the cases ruled out in this section, see Chapter 7, Advanced Schema Evolution, in the *Advanced C++ API User Guide*.

Use osbackup first

Running the `ossevol` utility changes the physical structure of your database. Consequently, you should back up your database before you run the `ossevol` utility and work on a copy of the database.

You can restart `ossevol` if it is interrupted (by a system crash, for example) and it will resume, using a checkpoint stored in the work database.

Syntax

```
ossevol workdb new_schema.adb evolvedb ... options
```

Arguments

`workdb`

Specifies the name of a scratch file that `ossevol` uses as a workspace; you can delete this file after `ossevol` has finished successfully.

`new_schema.adb`

Specifies the name of the revised and regenerated application schema database.

`evolvedb`

Specifies the name of the copy of the database to be evolved, which `ossevol` modifies to become the evolved database. The names of multiple affiliated databases that share the same schema can be specified here.

Options

Numeric values can be in decimal or hexadecimal format. Hexadecimal values must be prefixed by the characters 0x.

`-address_space_release_interval` *`interval`*

Controls the frequency of address-space releases, in units of bytes. Specify a smaller value for more frequent releases, a larger value for less frequent. The default value is displayed when you run `ossevol with no arguments.`

This option is useful when you want to prevent schema evolution from running out of address space while executing transformer functions.

`-classes_to_be_removed` *`class_name`* `...`

Specifies the names of the classes to be removed. Separate class names with a space.

`-dll_schema` *`pathname`* `...`

Specifies the paths of component schema databases to be used in addition to the new application schema. Use spaces between multiple path names.

`-drop_obsolete_indexes { yes | no }`

Specifies whether obsolete indexes encountered in the course of the evolution should be dropped. When this option is set to `no`, the `ossevol` utility stops with

an error if it encounters an obsolete index. The default is `no`, obsolete indexes are not dropped.

`-explanation_level` *`n`*

Specifies the level of debugging information output by `ossevol`. The argument *`n`* must be an integer between `0` and `4`, which have the following meanings:

- `0` — Do not output debugging information (default).
- `1` — Output phase-level information for each cluster, segment, and database that is evolved.
- `2` — Output information about all types in the database.
- `3` — Output information about objects with transformers.
- `4` — Output all possible debugging information.

Note that the levels of information are accumulative: each level includes all information at the previous levels.

`-malloc_size` *`size`*

Specifies the malloc size in KB for the relocation map. The default is 1024. This option is rarely used.

`-maplet_size` *`size`*

Specifies the maplet size in bytes of the relocation map. The default is 64. This option is rarely used and is dangerous.

`-max_bytes_wasted_to_avoid_page_boundary` *`size`*

Specifies that you want schema evolution to avoid placing objects across page boundaries in the evolved database. This applies to objects that the utility moves because they have changed or because other objects have changed. When schema evolution avoids a page boundary, the utility inserts a free region and leaves some bytes on the page unused.

Replace *`size`* with the maximum number of bytes that it is okay to leave unused on a page. When the utility would leave more than *`size`* bytes free in order to avoid allocating an object across a page boundary, it does not avoid the page boundary. You can specify a value of `0` through `4095` for size. Avalue of `0` disables this feature. The default is that schema evolution does not avoid placing objects across page boundaries.

Note: In the context of this option, all pages are server pages, which are always 4K in size. It does not matter what platform the server is running on.

`-maximum_cluster_size` *`max_size_bytes`*

This specifies that clusters larger than the size represented by *`max_size_bytes`* will be split into multiple clusters. The value of *`max_size_bytes`* is rounded up to the next multiple of 64 KB. The default is to split any cluster that exceeds 1600 MB.

Note that some ObjectStore collections manage an internal cluster that will not be split because the internal clusters maintain data structures that cannot be split

across multiple clusters. Examples of collections that maintain internal clusters are `os_set`, `os_Set`, `os_bag`, `os_Bag`, `os_Dictionary`, and indexes.

`-memory` *`size`*

Specifies the maximum memory size in MB to use for the pointer relocation map. The default is half the main memory or half the virtual memory, whichever is smaller.

`-prefetch` *`size`*

Specifies the maximum size in KB of data to prefetch. The default is 1024 and provides good performance on most hardware.

`-resolve_ambiguous_void_pointers { yes | no }`

Resolves ambiguous void pointers to the outermost enclosing collocated object (`yes`). A `void` pointer is ambiguous if schema evolution cannot determine whether it points to an object or to the first data member within the object. When this option is set to `no`, the `ossevol` utility stops with an error if it encounters an ambiguous `void*` pointer. The default (`no`) is not to resolve ambiguous `void` pointers.

`-secondary_psr_size` *`size`*

Specifies the size in MB of the secondary sessions' persistent storage region (PSR). The default is `16`. Increase this over the default in the unlikely event that you run out of address space (for example, if you have an extremely large schema).

`-task_list` *`filename`*

Specifies that the `ossevol` utility should produce a task list and place it in the file specified by *`filename`*. Use a hyphen (`-`) for `stdout`. When you specify this option, ObjectStore does not perform schema evolution.

The task list consists of pseudofunction definitions that indicate the way the migrated instances of each modified class would be initialized. This allows you to verify the results of a schema change before you migrate the data.

`-threads` *`number_threads`*

Specifies the number of threads to use. The default is 1.5 times the number of CPUs in the system (rounded up). The default provides good performance on most hardware if there is no other significant load on the system.

You may need to decrease the number of threads when running on a system with many processors but a small persistent storage region (PSR) size. Using too many threads can cause the PSR size for the main thread to be too small, causing address space exhaustion. It is acceptable to restart an interrupted schema evolution while specifying a different number of threads. You can increase the address space available to the main thread by decreasing the number of threads, by decreasing the `address_space_release_interval`, or by setting the environment variable `OS_AS_SIZE` to a larger number.

Description

Changes can include

You can use the `ossevol` utility to evolve the following changes:

- Adding or deleting the first or last virtual member function
- Reordering data members

Changes cannot include

You cannot use the `ossevol` utility to evolve changes that include

- User-defined transformer functions
- Any exception-handling code not handled by `ossevol` command-line options
- A pointer-to-member that points to a member of a virtual base class
- Untranslatable pointer handlers

Changes might include

You might be able to use the `ossevol` utility to evolve the following changes. In each item, the information after the first sentence indicates reasons that the `ossevol` utility might not be able to perform the evolution.

- Deleting obsolete classes or instances. Deleting an object might require modification of data belonging to another object, such as a counter.
- Adding or deleting base classes. When adding base classes, you might want some data members to be initialized with a value other than 0. When deleting base classes, you might want another effect.
- Adding or deleting a data member (also indexable data members). When adding a data member, you might want it to be initialized to something other than `0`.
- Changing the type of a class member. Depending on the type you are changing to, there might be a requirement for some transformations that cannot be done automatically.

Changes not requiring schema evolution

These changes do not require schema evolution:

- Adding a new class. However, if you add a nested class on Windows platforms and the alignment of the enclosing class is smaller than the alignment of the nested class, schema evolution is required.
- Changing member access (private, protected, or public).
- Changing an integer data member to or from signed or unsigned.
- Changing a data member to `const` from non-`const`, and vice versa.
- Adding or deleting static data members.

Except on Windows, the following two changes do not require schema evolution. On Windows, these two changes require schema evolution in some cases. You receive a schema validation message when you access the database with the new schema. This indicates that schema evolution is required.

- Adding or deleting nonvirtual member functions
- Adding or deleting virtual member functions (other than the first or last)

Changes requiring transformer functions

These changes require application-specific transformer functions:

- Renaming a data member
- Initializing instances of a new data member from data members of existing instances

- Making a nonindexable data member indexable, and vice versa
- Adding a data member that is a C++ reference or a `const`
- Moving data members to or from a derived or base class
- Reinitializing user-defined structures containing values that are addresses of evolved instances

Schema protection

If you are developing an application and will run this utility on a protected schema database, ensure that the correct key is specified for the environment variables `OS_SCHEMA_KEY_LOW` and `OS_SCHEMA_KEY_HIGH`. If the correct key is not specified for these variables, the utility fails. ObjectStore signals

```
err_schema_key _CT_invalid_schema_key,
"<err-0025-0151> The schema is protected and the key provided did not
match the one in the schema."
```

If you deploy an application in which your end users will need to use the `ossevol` utility on databases protected by a schema key, you must supply them with an application that uses the ObjectStore schema key API to set the correct schema key and then `os_schema_evolution::evolve()`. End users need not know anything about the key.

For more information about protecting a database with schema keys, see Restricting Access with Schema Keys in the *C++ API User Guide*.

Schema evolution API

For complete information about the schema evolution API, see `os_schema_evolution` on page 331 in the *C++ API Reference* and in Chapter 7, Advanced Schema Evolution in the *Advanced C++ API User Guide*. For information about using the `ossevol` utility on an ObjectStore Java database while an application is attempting to access objects stored in the database, see the *Java API User Guide*, Chapter 6 (“Storing, Retrieving, and Updating Objects”), the section entitled “Caution about Retaining Unexported Objects”.

Very large databases

See *Advanced C++ API User Guide*, Evolving or Compacting Very Large Databases.

ossg

The `ossg` utility is the ObjectStore schema generator. See *Building C++ Interface Applications* for complete information about how to use `ossg` and for a variety of examples.

Syntax

The syntax for the schema generator utility, `ossg`, varies according to the kind of schema being generated.

Application or component:

```
ossg [compiler_options] [neutralizer_options] [-cpp_fixup]
[-E] [-final_asdb final_app_schema_db] [{-mrlcp | -mrscp}]
[-no_default_includes] [-no_weak_symbols]
[-rtdp {minimal | derived | full | maximal}]
[-store_member_functions] [-weak_symbols]
{-assf app_schema_source_file | -asof app_schema_object_file.obj}
-asdb app_schema.adb schema_source_file [lib_schema.ldb ...]
```

Library:

```
ossg [compiler_options] [neutralizer_options] [-cpp_fixup]
[-mrscp] [-no_default_includes] [-store_member_functions]
-lsdb lib_schema.ldb schema_source_file
```

Compilation:

```
ossg [compiler_options] [neutralizer_options] [-cpp_fixup]
[-mrscp] [-no_default_includes] [-store_member_functions]
-csdb comp_schema.cdb schema_source_file
```

Argument *schema_source_file*

Specifies the C++ source file that designates all the types you want to include in the schema. It should include all classes that the application uses in a persistent context.

This argument is almost always required; there is no default. The schema source file is not required when you use a compilation schema to generate an application schema.

Also, you can omit the schema source file if you are generating an application schema and you specify one or more library schemas that contain all persistent types that your application uses.

Options

compiler_options

Specifies any options that would be passed to the compiler if you were compiling a schema source file instead of generating schema from it. You should include any preprocessor options, such as `include` file paths and macro definitions, as well as compiler options that might affect object layout, such as packing options (for example, `/Zp4` for Visual C++).

If you specify the `/I`, `-I`, `/D`, or `-D` option, do not include a space between the option and the argument. For example, on Windows the following is correct:

```
ossg /I%OS_ROOTDIR%\include ...
```

On UNIX, do not specify the `-o` option on the `ossg` command line.

On Windows, do not specify `/Tp` on the `ossg` command line.

neutralizer_options

Include any of the options described in Neutralization Options on page 235. These options allow you to neutralize a schema for a heterogeneous application. You can include them in any order.

`-cpp_fixup`

Allows preprocessor output to contain spaces inside C++ tokens. Specify this option if your preprocessor inserts a space between consecutive characters that form C++ tokens. For example, if your preprocessor changes `::` to `: :`, you can specify this option so that the schema generator allows the inserted space and correctly reads the preprocessor output.

It is possible to generate an application schema from a compilation schema and library schemas. In this case, you do not need this option because there is no source code input to the schema generator, which means that the preprocessor is not involved.

If this option is not specified, the default is that `ossg` does not allow a space in a C++ token such as `::` or `.*`.

`-E`

Causes `ossg` to preprocess the schema source file and send the preprocessed output to standard output. This option is useful for debugging `ossg` parsing problems because it allows you to see the results of any preprocessing. It is also useful when reporting `ossg` problems to Technical Support representatives because it allows the problem to be reproduced by Technical Support without the need to package your application's `include` files.

When you specify the `-E` option, you cannot specify schema databases on the same command line. You can specify only the schema source file and preprocessor switches.

`-final_asdb` *final_app_schema_db*

Specifies a location for the application schema database that is different from the location you specify with the `-asdb` option. The schema generator writes the location you specify with the `-final_asdb` option into the application schema

source file (application schema object file for Visual C++). Use this option when you cannot specify the desired location with the `-asdb` option. The `-asdb` option is still required. The location it specifies is where the schema generator places the application schema.

This option is useful when you plan to store the application schema database as a derived object in a ClearCase Versioned Object. The schema generator cannot place the application schema database directly in a ClearCase VOB. If you specify the `-final_asdb` option with the desired location, you avoid the need to run the `ossetasp` utility, which patches an executable so that it looks for its application schema in a database that you specify.

After you run `ossg` with the `-final_asdb` option, remember to move the application schema to the database you specify with `-final_asdb`.

You must specify an absolute path name with `-final_asdb`.

If this option is not specified, the default is that the schema generator writes the path name that you specify with `-asdb` in the application schema source file (object file for Visual C++).

`{-mrlcp` or `-make_reachable_library_classes_persistent}`

Causes every class in the application schema that is reachable from a persistently marked class to be persistently allocatable and accessible.

This option is supplied for compatibility purposes only. The use of the `-mrlcp` option is discouraged. Specify `-mrscp` instead.

When you specify the `-mrlcp` option, you cannot neutralize the schema for use with a heterogeneous application. If you are building a heterogeneous application, you must either mark every persistent class in the schema source file or specify the `-mrscp` option.

If you do not mark any types in the schema source file and you specify `-mrlcp` when you run `ossg`, the application schema does not include any types. You must mark at least one type for there to be any reachable types.

If this option is not specified, the default is that only marked classes are persistently allocatable and accessible.

See also Determining the Types in a Schema in *Building C++ Interface Applications*.

`{-mrscp` or `-make_reachable_source_classes_persistent}`

Causes every class that is both defined in the schema source file and reachable from a persistently marked class to be persistently allocatable and accessible.

The difference between `-mrscp` and `-mrlcp` is that when you specify `-mrscp`, it applies to the schema when `ossg` is translating from source to schema. This allows the schema generator to recognize those types you plan to allocate persistently. The `-mrlcp` option applies to the application schema after the merging of constituent schemas.

The benefit of specifying the `-mrscp` option is that it allows you to perform a persistent `new` for a type that you did not explicitly mark in the schema source file. The drawback is greater execution time and executable size overhead.

If you do not mark any types in the schema source file and you specify `-mrscp` when you run `ossg`, then the application or compilation schema does not include any types. You must mark at least one type for there to be any reachable types.

If this option is not specified, the default is that only marked classes are persistently allocatable and accessible.

See also Determining the Types in a Schema in *Building C++ Interface Applications*.

`{-no_default_includes` or `-I-}`

When you specify this option, `ossg` does not automatically specify any `include` directories to the C++ preprocessor. However, the preprocessor can have default `include` directories built into it and `ossg` does check these built-in directories. Typically, the preprocessor uses built-in `include` paths to find standard `include` files such as `stdio.h`. Except for these built-in directories, when you specify this option, you must explicitly specify directories that contain `included` files.

For example, on some UNIX systems, when you do not specify this option, the C++ preprocessor looks for `include` files in the `/usr/include` directory.

Note that if you want the schema generator to pass the ObjectStore `include` directory to the preprocessor as a directory for finding `included` files, you must always specify it. For example:

UNIX: `-I$OS_ROOTDIR/include`

Windows: `/I$(OS_ROOTDIR)\include`

Specifying `-I-` is the same as specifying `-no_default_includes`.

If this option is not specified, the default is that the preprocessor checks default directories for `included` files.

`-no_weak_symbols`

Disables mechanisms that suppress notification about missing vtbls. This option allows you to check whether any vtbl symbol referenced is undefined.

If you specify `-rtdp maximal -no_weak_symbols`, the linker provides messages about what is missing. You can use this information to determine the additional classes you need to mark. These missing symbols are only a hint about what you might consider marking. They might also be the result of a link line error.

This default option specifies that the schema generator notify you about missing vtbls. To change this behavior, specify the `-weak_symbols` option.

`{-rtdp` or `-runtime_dispatch} {minimal | derived | full | maximal}`

Specifies the set of classes for which the schema generator makes vtbls available.

`minimal` specifies marked classes, classes embedded in marked classes, and base classes of marked classes.

`derived` specifies the minimal set plus classes that derive from marked classes and classes embedded in the derived classes.

`full` specifies the derived set plus the transitive closure over base classes, derived classes, and classes that are the targets of pointers or references. The `full` specification does not include nested classes or enclosing classes unless they meet one of the previous criteria.

`maximal` specifies the full set plus nested types.

If this option is not specified, the default is `derived`.

`-suncc_550`

Use this option if you are using the Forte compiler on Solaris sol2c5 or sol64 platforms and encounter compiler errors relating to that compiler's use of the `_global`, `_hidden`, and `_symbolic` language extension keywords.

`-store_member_functions or -smf`

Causes `ossg` to create an instance of `os_member_function` for each member function in each class in the schema source file. It then puts these instances in the list of class members, which includes member types and member variables.

This option is useful when you intend to use the Metaobject Protocol (MOP) to inspect the member functions. If you are not planning to inspect member functions, you should not specify this option because it wastes disk space.

When you use this option, additions and deletions of member functions are schema changes and affect validation.

When you generate an application schema, you might specify a library or compilation schema. If you want to capture the member functions from the library or compilation schema, you must have specified the `-store_member_functions` option when you generated the library or compilation schema. You must also specify this option when you generate the application schema.

If this option is not specified, the default is that `ossg` generates a schema that includes member types and member variables, but not member functions.

`-weak_symbols`

Enables mechanisms that suppress notification about missing vtbls. This option overrides the default behavior described at `-no_weak_symbols`.

UNIX `-assf` *`app_schema_source_file`*

Windows `-asof` *`app_schema_object_file`*`.obj`

Specifies the name of the application schema source file or application schema object file to be produced by `ossg`. On UNIX platforms, use the `-assf` option; the schema generator will produce a source file that you must compile. On Windows, use the `-asof` option; the schema generator will produce the object file directly.

This option is required when generating an application schema. There is no default.

`-asdb` *`app_schema`*`.adb`

Specifies the name of the application schema database to be produced by `ossg`. If the schema database exists and is compatible with the type information in the input files, the database is not modified.

This path name must be local to a host running an ObjectStore server.

The path name should have the extension `.adb`. If you want to specify an existing application schema database with `ossg`, the application schema must have `.adb` as its extension.

This option is required when generating an application schema. There is no default.

`-csdb` *`comp_schema`*`.cdb`

Specifies the path of the compilation schema database to be generated by `ossg`. ObjectStore Technical Support suggests that the path name end in `.cdb`.

This path name must be local to a host running an ObjectStore server.

This option is required when generating a compilation schema. There is no default.

`-lsdb` *`lib_schema`*`.ldb`

Specifies the path of the library schema database to be generated by `ossg`. The path name must end in `.ldb`.

This path name must be local to a host running an ObjectStore server.

This option is required when generating a library schema. There is no default.

`lib_schema``.ldb ...`

Specifies the path of a library schema database. The name must end in `.ldb`. The library schema database is one that you previously created with `ossg`.

The schema generator reads schema information from the library schema database specified and modifies the application schema database to include the library schema information. You can specify zero or more library schema databases.

If this option is not specified, the default is that library schemas are not included.

Neutralization Options

`-arch` *`set`*

The schema that is generated or updated is neutralized to be compatible with the architectures in the specified set. Applications running on these architectures can access a database that has the schema.

This option is required when you are neutralizing schema. There is no default. You can specify one of the following values of *`set`*:

- `all32` — All 32-bit architectures
- `all64` — All 64-bit architectures
- One of the versioned architecture set supported by `ossg`; see Versioned Architecture Sets in Chapter 5 of *Building C++ Interface Applications*.
- Forced-order architecture set — for use when you must consider allocation order when generating schema for applications that use virtual base classes. For more information, see Neutralizing the Allocation Order of Virtual Base Classes in Chapter 5 of *Building C++ Interface Applications*.
- A user-defined set specified by the `OS_USER_ARCH_SET` environment variable; see `OS_USER_ARCH_SET`.

You can use `ossg`'s `-showsets` option to list the platforms supported by any of the ObjectStore-defined architecture sets.

Note

When neutralizing schema, you might have to consider allocation order when generating schema for applications that use virtual base classes. For more information, see Neutralizing the Allocation Order of Virtual Base Classes.

`-ignore_vbo`

Suppress any warnings about differences in the allocation order of virtual base classes. For more information, see Neutralizing the Allocation Order of Virtual Base Classes in Chapter 5 of *Building C++ Interface Applications*.

`-nout` *`filename`* or `-neutral_info_output` *`filename`*

Indicates the name of the file to which neutralization instructions are directed.

If this option is not specified, the default is that the schema generator sends output to `stderr`.

`-nor[eorg]`

Prevents the schema generator from instructing you to reorganize your code as part of neutralization.

This is useful for minimizing changes outside your header file, working with unfamiliar classes, or simply padding formats.

When you include `-noreorg`, your application might not make the best use of its space. In fact, it is seldom possible to neutralize a schema without reorganizing classes.

When you use virtual base classes, it is very unlikely that you can neutralize your schema when you include this option.

If this option is not specified, the default is that the schema generator provides reorganization instructions.

`{{-pad_maximal` or `-padm} | {-pad_consistent` or `-padc}}`

Indicates the type of padding requested.

`-pad_maximal` or `-padm` indicates that maximal padding should be done for any ObjectStore-supported architecture. This means all padding, even padding that the various compilers would add implicitly.

`-pad_consistent` or `-padc` indicates that padding should be done only if required to generate a consistent layout for the specified architectures.

If this option is not specified, the default is `-padc`.

`-ptn` *`name size`* or `-portable_type_name` *`name size`*

Specifies that a type, *`name`*, has a given *`size`*, in bytes, on all architectures in the architecture set specified with `-arch`. Usually, *`name`* is conditionally defined in a `typedef` and it must be an integer type.

The schema generator does not flatten portable type names that are used as template type actuals. This yields identical class names across platforms. For example, consider the following:

```
#ifdef WIN32
typedef __int64 some_8byte_integer;
#else
typedef long long some_8byte_integer;
#endif

template<class T> class foo {};

OS_MARK_SCHEMA_TYPE(foo<some_8byte_integer>);
```

This installs `foo<some_8byte_integer>`, which all platforms can access, into the database schema.

`-schema_options` *`option_file`* or `-sopt` *`option_file`*

Specifies a file in which you list compiler options being used on platforms other than the current platform. The options in this file usually override the default layout of objects, so it is important for the schema generator to take them into account. See Listing Nondefault Object Layout Compiler Options in Chapter 5 of *Building C++ Interface Applications* for details about the content of the option file.

Optional. There is no default.

`{{-show_difference` or `-showd} | {-show_whole` or `-showw}}`

Indicates the description level of the schema neutralization instructions.

If this option is not specified, the default is `-show_whole`.

`-showsets`

Lists all architecture sets that can be specified as arguments to the `-arch` option. Included in the listing are the names of all platforms that are members of each set.

`{-wchar_t2 | wchar_t4}`

Specifies the size of `wchar_t` to be either 2 or 4 bytes.

This option is required if a schema includes `wchar_t`s.

ossize

The `ossize` utility displays the size of the specified database and the sizes of its segments and clusters.

Syntax

```
ossize [options] pathname
```

Argument

pathname

Specifies the file or rawfs database whose size you want to display.

Options

`-a`

Displays the details of all internal database structures.

`-A`

Displays access control information.

`-c`

Displays the type contents for each segment. Includes the counting of free spaces. Compare the `-0` option.

`-C`

Displays the type contents for the entire database. Includes the counting of free spaces.

`-f`

Displays a report on the fragmentation of the database specified by *pathname*. Sizes are reported in kilobytes. For more information about the `-f` option, see Database fragmentation on page 239.

`-L` *transaction_log_name*

For use with ObjectStore / Single only. When this option is specified, the named file is used for the transaction log file. If a file with the name *transaction_log_name* already exists, it must be a properly formed transaction log. By default, `ossize` uses a temporary file.

`-n` *segment_number*

Displays information only about the segment specified as *segment_number*. *segment_number* is a data segment number such as those displayed by the `-a` (`/INFO`) option. This option is useful with the `-o` (`/DEBUG`) and `-c` (`/SEGMENT`) options because it reduces the amount of output.

`-o` (the letter)

Displays a complete table of every object in the segment, showing its offset and size. The data in this table can be useful in debugging. Do not confuse this with the `-0` (zero) option, described later in this list.

`-sn`

Displays the type summaries by the number of instances of each type.

`-ss`

Displays the type summaries by the space used by the instances of each type. (This is the default.)

`-st`

Displays the type summaries alphabetically by type name.

`-0 (zero)`

Causes `ossize` to display the type contents for each segment, including the internal segment 0. Compare with the `-c` option.

Description

The `ossize` utility does not distinguish persistently allocated pointers (such as `new(db) thing*` or `new(db) thing*[100]`) as separate types. They are displayed together.

The `ossize` utility displays the comment for each segment that has a comment with a nonzero length. See `os_segment::set_comment()` in the *C++ API Reference*.

Schema protection

If you are developing an application and will be running this utility on a protected schema database, ensure that the correct key is specified for the environment variables `OS_SCHEMA_KEY_LOW` and `OS_SCHEMA_KEY_HIGH`. If the correct key is not specified for these variables, the utility fails. ObjectStore signals

```
err_schema_key _CT_invalid_schema_key,
"<err-0025-0151> The schema is protected and the key provided did not
match the one in the schema."
```

Database fragmentation

When the `ossize` utility is invoked with the `-f` option, it produces a statistical report on the fragmentation for three types of physical data structure within a database:

- *Clusters*, which refer to the disk space that holds the data (as opposed to metadata) for a cluster.
- *Metadata table directories*, which point to metadata table leaves in clusters that are large enough to need more than one leaf per table.
- *Metadata table leaves*, which contain per-page metadata such as free space, tags (object type information), and PRMs (pointer reference maps).

Of the three types, clusters are the most likely to suffer fragmentation.

For each type of physical data structure, `ossize` displays statistical information for those data structures. If any of the data structures is fragmented (that is, if at least one of them is not contained entirely within a single contiguous range of disk sectors), the report includes additional information for the fragments. Note that the report ignores any physical data structures that are not fragmented when computing this additional information. For example, the average number of fragments per cluster is not biased downward by the clusters that have only one fragment; it is really the average number of fragments per fragmented cluster.

The statistical information includes the minimum and maximum values encountered, the average value, and an estimate of the median value. The median is

a number such that half of the values are lower and the other half are higher. The median is not computed exactly but is usually within 20% of the true median. Because these distributions tend to be very skewed, with more small values than large values, the median is often much smaller than the average.

For an example report displayed by `ossize -f`, see the Examples section, below. For more information about database fragmentation, see Managing Database Fragmentation on page 44.

API

See `os_dbutil::ossize()` on page 185 in the *C++ API Reference*.

Examples

Rawfs database

```
$ ossize readers.db

Name: /h/handcuff/2/ebailey/release/library/ug2/readers.db
Size: 57344 bytes (56 Kbytes)
Created: Fri Feb  5 16:23:17 1999

There is 1 root:
Name: readers   Type: Reader

Affiliated databases:
/h/handcuff/2/ebailey/release/library/ug2/books.db

The schema is local.

There is 1 segment:

Data segment 2:
Size: 512 bytes (1 Kbytes)
Segment 2 has 1 cluster:
Cluster 0 size: 512 bytes (1 Kbytes)
```

-f example

The `-f` option causes `ossize` to report fragmentation statistics for the physical structure of the specified database. For example:

```
Database has 43 clusters, of which 15 (34%) are fragmented.
Cluster size min 0.5K max 2992K average 305.5K estimated median 8K
Fragments per cluster min 3 max 3944 average 1039 estimated median 23
Fragment size min 0.5K max 72K average 1K estimated median 0.5K

Database has 4 metadata table directories, of which 0 (0%) are
fragmented.
Directory size min 0.5K max 0.5K average 0.5K estimated median 0.5K

Database has 173 metadata table leaves, of which 0 (0%) are fragmented.
Leaf size min 0.5K max 6.5K average 2.5K estimated median 0.5K
```

This database contains many small clusters (median cluster size is 8K) as well as some larger clusters (maximum cluster size is almost 3 MB). Some clusters are extremely fragmented, with most fragments being only 0.5KB (the smallest possible size) and one cluster having 3,944 fragments.

ossvrchkpt

The `ossvrchkpt` utility performs a checkpoint for a specified server host. It ensures that all data is copied from the transaction log of the server host to the database or databases that were changed.

Note

If you are using ObjectStore / Single, you must us the `osprop` utility to perform a checkpoint; see `osprop` on page 198.

Syntax

```
ossvrchkpt hostname
```

Argument

`hostname`

Specifies the name of the host of the server whose log you want to propagate.

Description

This command does not return until the propagation is complete. It can return the following values:

Value	*Meaning*
`0`	Success.
`1`	There is an error when passing the command to the server.
`2`	The server is unable to complete the checkpoint.

Run this utility when you want to

- Ensure that a server is restarted as quickly as possible
- Use operating system file commands such as copy or move on a file database
- Reinstall or upgrade ObjectStore

API

See `os_dbutil::svr_checkpoint()` on page 191 in the *C++ API Reference*.

Example

```
ossvrchkpt hostess
```

In this example, data in the server transaction log on the host called `hostess` is copied to the databases that were modified.

ossvrclntkill

The `ossvrclntkill` utility disconnects a client thread on the server running on the specified host. This disconnects the client from the server and releases the client locks.

Syntax

There are two forms of the `ossvrclntkill` utility. The second form is supported for compatibility with earlier releases.

```
ossvrclntkill hostname [ -h client_host ] [ -p client_pid ]
    [ -n client_name ] [-a]
ossvrclntkill hostname client_pid
```

Argument

`hostname`

Specifies the name of the host of the server that is connected to the client process being disconnected.

Options

`-h client_host`

Specifies the name of the host of the client being disconnected, as determined with `ossvrstat`.

`-p client_pid`

Specifies an unsigned number that is the process ID of the client process being disconnected.

`-n client_name`

Specifies the name of the client process being disconnected. This name is set by `objectstore::set_client_name()`.

`-a`

Specifies that all clients matching the specified criteria should be disconnected.

Description

Run the `ossvrclntkill` utility on the servers connected to the client that you want to kill. This utility returns an exit code of

- `0` when the utility succeeded
- `1` when it failed

You can use `ossvrstat` to determine the *`client_host`* and *`client_pid`*.

Use the `ossvrclntkill` utility when a client that no longer exists is still attached to the server. This can happen because of network failure or because the client process terminates abnormally. In most cases, the operating system disconnects the client from the servers gracefully, but some operating systems are not completely dependable in this regard.

You must specify one or more of `-h`, `-p`, or `-n`. The `-a` option deletes all matching clients; otherwise, a unique match is required.

If the server's authentication is set to something other than `NONE` (authentication is `SYS` by default), the following rule applies:

Any user can disconnect clients that the user owns. If the `-a` option is used (kill all clients matching the given search pattern), the user must own all matching processes; otherwise, authentication fails and no clients are killed.

Otherwise, no authentication is required.

API

See `os_dbutil::svr_client_kill()` on page 191 in the *C++ API Reference*.

Example

```
ossvrclntkill hostess -h cupcake -a
```

In this example, all clients on `cupcake` that are attached to the server on `hostess` are disconnected.

ossvrdebug

The `ossvrdebug` utility turns on debug output from an ObjectStore server.

Syntax

```
ossvrdebug hostname n
```

Options

`hostname`

Server host to be debugged.

`n`

Number that specifies the trace level of the server. Specify an integer from `0` through `9`. The larger the number, the more information ObjectStore provides. Specifying `0` turns the debug trace level off.

Description

The `ossvrdebug` utility sets the server debug trace level of the server. Using this command is equivalent to starting the server with the `-d` *`n`* command-line option. After the server receives the message from `ossvrdebug` to enable debugging, it sends trace output to `/tmp/oss_out` (UNIX) and `%OS_TMPDIR%\OSSERVER.TXT` (Windows).

API

See `os_dbutil::svr_debug()` on page 192 in the *C++ API Reference*.

Example

```
ossvrdebug kellen 5
```

This example sets the server debug trace level of the server `kellen` to `5`.

ossvrping

The `ossvrping` utility reports whether a server is running on the specified host.

Syntax

```
ossvrping [-v] [hostname]
```

Argument

`hostname`

Specifies the name of the host on which you want to know whether a server is running.

Option

`-v`

Indicates that you want more information when a server is not running on the specified host.

Exit codes

The `ossvrping` utility has the following exit codes:

Exit code	*Meaning*
0	Server is running and is not a failover server.
1	No server is running.
2	No such host.
3	Internal error; report problem to Technical Support.
4	Server is running as the failover secondary server.
5	Server is running as the failover primary server.

For information about failover servers, see Failover on page 285.

Description

If you are having problems with a server, first run `ossvrping` to see if the server is running.

If you do not specify a host, the default is the local host.

API

See `os_dbutil::svr_ping()` on page 193 in the *C++ API Reference*.

Example

```
ossvrping elvis
The ObjectStore Server on host elvis is alive.
```

ossvrshtd

The `ossvrshtd` utility shuts down the server running on the specified host. Shutdown occurs regardless of whether clients are connected to the server.

Syntax

```
ossvrshtd [-f] hostname
```

Argument

`hostname`

Specifies the name of the host of the server that you want to shut down.

Option

`-f`

Specifies that shutdown should occur without user confirmation. If you do not specify this option, ObjectStore prompts you to confirm that you really want to shut down the server.

Description

Before shutting down the server, run the `ossvrstat` utility to determine whether there are clients using the server. If there are, notify them to exit. Note that `ossvrshtd` initiates the shutdown process; it does not wait for shutdown to complete before returning.

Clearing the log

Shutting down the server automatically propagates everything in the transaction log.

If any clients are connected to a nonfailover server when you shut it down, the next time those clients try to contact the server, they receive the `err_broken_server_connection` exception. The client can call `os_server::reconnect()` to try to reconnect. If clients are connected to a failover server when you shut it down, they will reconnect and retry automatically, as directed by the locator file. For more information, see Locator File: FAILOVER_SERVER Declaration on page 291.

When needed

ObjectStore needs to be shut down when you

- Boot or halt the host
- Modify server password or parameters
- Reallocate the server transaction log
- Add a partition to the rawfs

UNIX

If the server's authentication is set to `NONE` (authorization is `SYS` by default), you must be the user ID that owns the running server process, or the superuser, to run this utility.

If the server's authentication is set to something other than `NONE`, `ossvrshtd` must be run as `root`.

Cluster operating system

You cannot use the `ossvrshtd` utility to shut down a server that is running on a node of the Sun Clusters operating system. The cluster operating system will regard the attempt to shutdown the server with the `ossvrshtd` utility as a server failure and

immediately start the server up again. If you want to take the server down for maintenance, you need to tell the cluster operating system to disable the resource by using the Sun Management Center GUI or by running the `stop_osserver` script that we supply in the EXLNossvr package.

Windows

On Windows, you can shut down the server by using the Service Control Manager. Click the Services icon in the Control Panel or issue the command `net stop "ObjectStore Server R6.0"`.

Starting a server

For instructions for starting a server, see the chapter in this book for your platform.

API

See `os_dbutil::svr_shutdown()` on page 193 in the *C++ API Reference*.

Example

```
ossvrshtd hostess
Are you sure that you wish to shut down the server
on host hostess (yes/no) [no]: yes
```

ossvrstat

The `ossvrstat` utility displays settings of server parameters, server use meters, and information for each client connected to the server running on the specified host.

Syntax

```
ossvrstat hostname [options]
```

Argument

`hostname`

Specifies the name of the host of the server for which you want information.

Options

`-meters`

Displays performance meters for the specified server.

`-clients`

Displays the state of each client connected to the specified server and shows those clients, if any, that are contending for locks.

`-databases`

Displays information about databases that were opened by each client connected to this server. For more information about this option, see Open Databases Information on page 252.

`-parameters`

Displays server parameter values. The following parameters are not displayed if they are not enabled: `Allow NFS Locks`, `Allow Remote Database Access`, and `Host Access List`.

`-rusage`

UNIX only: displays server process information.

`-log`

Displays information about the server transaction log on the specified server. For more information about this option, see Server Transaction Log Information on page 253.

Description

Specifying both `-meters` and `-clients` displays the use meters for all clients, as well as the information described above.

ObjectStore identifies each client by host name and then displays the program name (if there is one) with the process ID on the client host. Program names are set with `objectstore::set_client_name()`. When there is no program name, ObjectStore displays `default_client_name`.

API

See `os_dbutil::svr_stat()` on page 193 in the *C++ API Reference*.

Numbers are relative

The table on the next few pages describes the `ossvrstat` meters. The utility supplies high and low numbers for each. However, the numbers are entirely relative to your

application. For high and low to have meaning, run `ossvrstat` to determine a baseline.

Default invocation

If you invoke the `ossvrstat` utility without any options, it displays performance meters, client information, and server parameters for the specified server.

The following table describes the performance meters that `ossvrstat` displays.

Meter	*Description*
`Current log size`	Number of sectors in the transaction log. This meter appears in the middle of the list of server parameters because it is most useful when you are determining the way to adjust other server parameters.
`Messages received`	Number of messages the server has received from clients. A message can be a request for an action such as opening a database, fetching a database page, updating a database, committing a transaction, aborting a transaction, closing a database, and other related actions. The number indicates how often clients are communicating with the server. When the number is low, the demand on the server is less.
`Callback messages sent`	Number of callback messages the server sent to clients. A callback message is the message a server sends to clientA when clientB requests data on a page that is cached by clientA. When this number is high, it means that an application is often modifying data that other clients want to read or modify. This might indicate that the program is poorly designed.
`Callback sectors`	Number of 512-byte sectors that have been called back in callback messages. This is not necessarily the same as the number of sectors for which pages actually have been shared or released. Also, the server might send many callback messages but they might not be for a large number of sectors. Usually, callbacks are for pages (4 KB or 8KB on most machines). Sometimes the server calls back larger chunks.
`Succeeded sectors`	Number of sectors for which the server sent a callback message and the client that had the page(s) cached (clientA) either shared the page(s) with clientB or relinquished the page(s) to clientB. If `Callback sectors` is comparable to `Succeeded sectors`, you know that clients are not waiting too long. If `Succeeded sectors` is much smaller, more clients are being locked out of data they need.
`KB read`	Number of kilobytes of data that the server sent to clients to read. Monitor this statistic to help determine whether you need to enlarge client cache files. Compare kilobytes read for a given client with the number of commits and the size of the client cache file.
`KB written`	Number of kilobytes of modified data that the clients sent to the server. Written data is data involved in a commit. It must be buffered, and it is logged if it cannot go directly to a database because it is not being written past the current end of a cluster. When analysis is concerned with the number of transactions per second, the number of kilobytes written is an important factor.
`Commits`	Number of committed transactions that the server knows about. If a client does not modify data during a transaction, the client might not inform the server that a transaction was committed. You can use this number to estimate the number of transactions per second.

Meter	*Description*
`Readonly commits`	Number of committed transactions that the server determined not to have involved any data changes. Typically, the client does not inform the server about such commits, so this number should be low. An example of this is when the client releases ownership needed by another client. In this case, the client sometimes performs a commit, even on a read-only transaction. Read-only commits are like simple aborts; the cost is near zero.
`Aborts`	Number of aborted transactions that the server knows about. If the client has not sent any messages to the server, the client can abort a transaction without informing the server. Most applications abort transactions only because of lock conflicts. In this case, you can use this number to determine the number of conflicts.
`Two-phase transactions`	Number of committed transactions that involved changes to databases on more than one server. Typically, one server is involved in a commit. A two-phase commit requires additional overhead, so it is useful to know how often it is happening.
`Lock timeouts`	Number of times all clients fail to obtain a lock because a lock timeout time is set on the client and the lock needed was not released before the lock timeout elapsed.
`Lock waits`	Number of times all clients had to wait to obtain a lock on a page because a lock by another client was already in place. The utility also provides the average time that a client waits for a lock. This appears in parentheses next to `Lock waits`. ObjectStore divides the total time waiting for locks by the number of lock waits.
`Deadlocks`	Number of times the server chose a client to be a deadlock victim and notified it that it had to abort a particular transaction so that other clients could complete their transactions. If you specify `-clients` when you run `ossvrstat`, the utility displays information about those clients that are waiting for locks and those clients that currently have those locks.
`Message buffer waits`	Number of times a message from a client to the server must wait to use a message buffer. The `N Message Buffers` server parameter specifies the number of message buffers that the server uses to communicate with clients. If the number of `Message buffer waits` is high, consider increasing the value specified for `N Message Buffers`. See `N Message Buffers`.
`Notifies sent`	Number of notification messages the server sent to all cache managers for delivery to clients on that cache manager's host. When the values for `Notifies sent` and `Notifies received` are both `0`, ObjectStore does not print information for these two meters.
`Notifies received`	Number of notification messages the server received from all clients. The server sends these messages to the cache manager on the host of the client that the message is for. When the values for `Notifies sent` and `Notifies received` are both `0`, ObjectStore does not print information for these two meters.
`Log records`	Number of records written to a log record segment of the transaction log. Each committed transaction writes a record to the log. This is a throughput number. Space in the log is continually reused.

Meter	*Description*
`Record segment switches`	Number of times the server switches from writing commit records in one log record segment to writing commit records in the other log record segment. To switch segments, the server must ensure that all changes that are recorded in the log record segment that is being switched to have been propagated to the databases. When the server needs to switch log record segments, if everything is not propagated, the server forces the propagations to happen quickly. Too large a number here indicates that the log record segments are not big enough. You can improve performance by increasing the log record segment initial size. See `Log Record Segment Initial Size`.
`Flush data`	Number of flushes to disk of data that was in the data segment of the transaction log. A flush ensures that the data is on the disk. It does not free the space the data occupies in the log. ObjectStore determines when to flush data.
`Flush records`	Number of flushes to disk of records that were in a log record segment of the transaction log. A flush ensures that the changes are on the disk. It does not free the space the records occupy in the log. ObjectStore determines when to flush records.
`KB data`	Number of kilobytes of data that the server wrote to the data segment of the transaction log. When this number is high, it means one of the following: • The client is returning a lot of data to the server before transactions commit. Data returned before commit always goes directly to the database or the log data segment. • More data is being returned at commit than fits in the log record buffer and, therefore, the server wrote it to the log data segment. This is a throughput number. Space in the log is continually reused.
`KB records`	Number of kilobytes of records that the server wrote to a log record segment of the transaction log. Each committed transaction writes a record to the log. This is a throughput number. Space in the log is reused continually.
`KB propagated`	Number of kilobytes of committed data that was propagated from the transaction log to the database it belongs in. The server performs propagation in small chunks that do not interfere with client activity. Propagation can include writing to databases, flushing data and records that are in the log and, sometimes, reading from the log data segment. After propagation, the space that the propagated data and records occupied in the log becomes available for new log entries. The number of kilobytes propagated can be smaller than the number of kilobytes written if the same data is written multiple times. ObjectStore propagates the last modification and discards earlier modifications.

Meter	*Description*
KB direct	Number of kilobytes of uncommitted data that the server stored directly in databases. A high number here is good because it means that the data did not have the overhead of going through the log. An efficient database-loading application should show a value of KB direct much higher than the values of KB data, KB records, and KB propagated. The server stores uncommitted data in the database when an application tries to write data past the end of the database cluster in which it needs to be stored.
Propagations	Number of times the server propagated data from the log to databases. The server moves small chunks of data each time it performs propagation. Note that this meter increments multiple times per propagation. If the number of propagations per second is high, the server is probably forced to propagate for one or more of the following reasons: • The server needs to switch log record segments but has not finished propagating changes in the next segment. In this case, increase the log record segment size. See Log Record Segment Initial Size. • The log data segment is full. Check KB data to see if that is high in relation to the size of the log data segment. If it is, increase the log data segment size. See Log Data Segment Initial Size. • The setting for the server parameter Propagation Buffer Size or Max Data Propagation Threshold might not be high enough. Check KB written to see if this number, when looked at as kilobytes per second, is high in relation to the parameters. If it is, increase the settings for the parameters if this is appropriate for the amount of physical memory on the server. For information about specifying the amount of data moved in one propagation, see Max Data Propagation Per Propagate.

Open Databases Information

Specifying the -databases option displays information about databases open by each client for the specified server. The information is displayed in the following format:

- First column is the ObjectStore release of the database: Release 5.x or blank.
- Second column is open mode (WRite, ReaD, or MVcc) and elapsed time since this client opened this database.
- Third column is cache state (CW if any pages cached for write, CR if any pages cached for read).
- Fourth column is lock state (WL if any write locks held, RL if any read locks held).
- Fifth column is transaction state (TW if written in current transaction, TR if read in current transaction, T if there is a current transaction but it has not touched this database as far as the server knows, blank if the server is not aware of any transaction) followed by C if there is an MVCC conflict.

- Sixth column is AIO thread number for the database (blank for rawfs).
- Finally, the database pathname.

For an example of output when `ossvrstat` is invoked with the `-databases` option, see -databases option on page 256.

Server Transaction Log Information

Specifying the `-log` option displays information about the server transaction log for the specified server. The information displayed consists of the following:

- The size of the transaction log and its components
- Metering information collected over various intervals of time
- Any databases for which propagation has been delayed due to MVCC access or backup and the amount of time since they were last allowed to be propagated
- Server parameters relevant to the transaction log and the values of related or derived parameters of the log itself
- AIO thread number for the transaction log, if any

Examples

Here is the output from `ossvrstat` when invoked without any options:

```
ossvrstat manacles
ObjectStore Release 6.1 Database Server
Client/Server protocol version 1.70

Allow Remote Database Access:  Yes
Remote Database Grow Reserve:  16MB
Allow Shared Communications:  Yes
Authentication Required:  SYS, DES, Name Password
Cache Manager Ping Time:  300 seconds
Cache Manager Ping Time In Transaction:  300 seconds
DB Expiration Time:  300 seconds
Deadlock Victim:  Work
Direct To Segment Threshold:  128 sectors (64KB)
Current Log Size:  92192 sectors (46096KB)
Log Data Segment Growth Increment:  2048 sectors (1MB)
Log Data Segment Initial Size:  81920 sectors (40MB)
Log Record Segment Buffer Size:  1024 sectors (512KB)
Log Record Segment Growth Increment:  512 sectors (256KB)
Log Record Segment Initial Size:  10240 sectors (5MB)
Max AIO Threads:  3
Max Connect Memory Usage:  unlimited
Max Data Propagation Per Propagate:  512 sectors (256KB)
Max Data Propagation Threshold:  81920 sectors (40MB)
Max Memory Usage:  unlimited
Max Two Phase Delay:  30 seconds
Message Buffer Size:  512 sectors (256KB)
N Message Buffers:  4
Notification Retry Time:  60 seconds
Preferred Network Receive Buffer Size:  16384 bytes
Preferred Network Send Buffer Size:  16384 bytes
Propagation Sleep Time:  60 seconds
Propagation Buffer Size:  81920 sectors (40MB)
```

```
RPC timeout:  60 seconds

Server Machine Usage:
  User time:  1.3 secs
  System time:  3.0 secs
  Max. Res. Set Size:  944
  Page Reclaims:  0
  Page Faults:  69
  Swaps:  0
  Block Input Operations:  2
  Block Output Operations:  526
  Signals Received:  744
  Voluntary Context Switches:  17894
  Involuntary Context Switches:  1237

Server Meters:
Total since server start up:
    Client Meters:
             280 messages received  0 callback messages sent
               0 callback sectors  0 succeeded sectors
             204 KB read  2752 KB written
              13 commits  0 readonly commits
              45 aborts  0 two phase transactions
               0 lock timeouts  0 lock waits
               0 deadlocks  0 message buffer waits
    Log Meters:
              62 log records  14 record segment switches
             116 flush data  103 flush records
               0 KB data  51 KB records
              13 KB propagated  1762 KB direct
             130 propagations

Total over past 60 minute(s):
    Client Meters:
              67 messages received  0 callback messages sent
               0 callback sectors  0 succeeded sectors
              31 KB read  744 KB written
               4 commits  0 readonly commits
              11 aborts  0 two phase transactions
               0 lock timeouts  0 lock waits
               0 deadlocks  0 message buffer waits
    Log Meters:
              16 log records  4 record segment switches
              32 flush data  28 flush records
               0 KB data  14 KB records
               4 KB propagated  436 KB direct
              40 propagations

Total over past 10 minute(s):
    Client Meters:
              14 messages received  0 callback messages sent
               0 callback   0 succeeded sectors
               0 KB read  186 KB written
               1 commits  0 readonly commits
               3 aborts  0 two phase transactions
               0 lock timeouts  0 lock waits
               0 deadlocks  0 message buffer waits
    Log Meters:
               4 log records  1 record segment switches
               8 flush data  7 flush records
```

```
                0 KB data  3 KB records
                1 KB propagated  109 KB direct
               10 propagations

Total over past 1 minute(s):
    Client Meters:
                0 messages received  0 callback messages sent
                0 callback sectors  0 succeeded sectors
                0 KB read  0 KB written
                0 commits  0 readonly commits
                0 aborts  0 two phase transactions
                0 lock timeouts  0 lock waits
                0 deadlocks  0 message buffer waits
    Log Meters:
                0 log records  1 record segment switches
                2 flush data  2 flush records
                0 KB data  0 KB records
                0 KB propagated  0 KB direct
                0 propagations

No active clients
```

Server machine usage

On UNIX systems, the `ossvrstat` output under the `Server Machine Usage` heading is provided by the `getrusage` utility. The output varies according to the platform on which the server is running. For information about what the output categories mean, see the manual page for `getrusage` on the server machine.

On non-UNIX platforms, the server fills in zeros for these output categories, which indicates that the measurement is not available on that platform.

Active clients

When there are active clients, the `ossvrstat` utility also displays a message such as the following:

```
Client connections awaiting a client message:
Client #3 (atiq/26896/(unknown))
  priority=0x8000, duration=4652 seconds, work=0, no
  transaction on server
Client #5 (nanook/1346/(unknown))   priority=0x8000,
  duration=2 seconds, work=2, transaction in progress
Client #7 (yukiko/14916/(unknown)) priority=0x8000,
  duration=136 seconds, work=0, no transaction on server
```

This is a list of the clients that have initiated a connection to the server. In the previous example, the server is waiting for the next message from each client. Next to the client number, the information in parentheses indicates the

- Host name of the client machine.
- Process ID of the client process on the client machine.
- Name of the client process. If you did not use the API to give the client process a name, ObjectStore displays `(unknown)`.

The other information provided is as follows:

- `priority` – A hexadecimal number that indicates the priority assigned to this transaction with the `os_transaction::set_transaction_priority()` function. ObjectStore uses this to determine the victim if there is a deadlock. The transaction with the lower number is the victim.

- `duration` – The number of seconds since the last successful commit by the client.
- `work` – The amount of work done by the client, as measured by remote procedure calls to the server during the current transaction. Each message to the server counts as one work unit.
- `comment` – Indicates whether a transaction is in progress.

-databases option

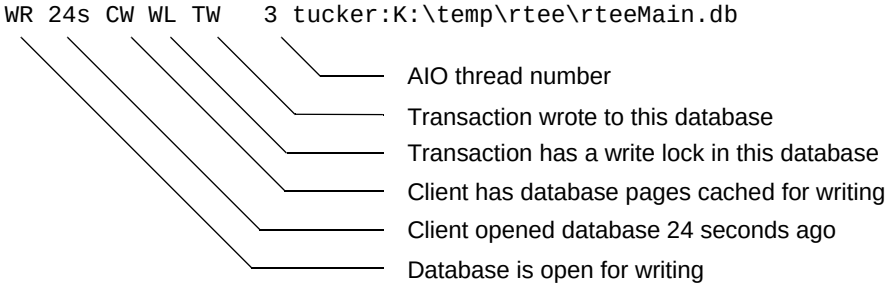
Here is the output from `ossvrstat` when invoked with the `-databases` option:

```
ossvrstat -databases tucker
ObjectStore Release 6.1 Database Server
Client/Server protocol version 1.70

Client Open Databases: (1 active client(s))
  Client #4 (tucker/1152/K:\rtee\coll.exe/RTEE - main session)
  WR  2s CW WL TW   1 tucker:K:\temp\rtee\rteeArchive0006.db
  WR  1s CW WL TW   2 tucker:K:\temp\rtee\rteeArchive0007.db
  WR  1s CW WL TW   3 tucker:K:\temp\rtee\rteeArchive0008.db
  WR  1s CW WL TW   1 tucker:K:\temp\rtee\rteeArchive0009.db
  WR 24s CW WL TW   3 tucker:K:\temp\rtee\rteeMain.db
  RD 24s CR RL TR   1 tucker:K:\RTEE\os\rtee\test\example.adb
  RD 24s CR RL TR   2 tucker:L:\build\debug\bin\os_coll.adb

Client connections awaiting a client message:
Client #4 (tucker/1152/K:\rtee\coll.exe/RTEE - main session)
  priority=0x8000, duration=27 seconds, work=2545, transaction
  in progress
```

For information about the format and type of information displayed by the `-databases` option, see Open Databases Information on page 252. The following decodes the information displayed for the 5th database (`rteeMain.db`) in the sample output:



Note that the MVCC conflict column, which appears just before the AIO thread number, is blank in this example because there are no MVCC clients.

ostest

The `ostest` utility indicates whether a path name meets a specified condition.

Syntax

```
ostest [options] pathname
```

Argument

`pathname`

The path name of a database or directory.

Options

`-d`

`pathname` is a rawfs directory.

`-f`

`pathname` is a rawfs database.

`-p`

`pathname` is a file path name.

`-r`

I (requestor) have read access to *`pathname`*.

`-s`

`pathname` is a database with a nonzero size.

`-w`

I (requestor) have write access to *`pathname`*.

Exit codes

The `ostest` utility has an exit code of `0` when the condition specified by the option is `true` and nonzero when the condition is `false`.

Description

You must specify one, and only one, option with the `ostest` utility.

UNIX only

When you invoke `ostest` on a UNIX system and specify a file database as the argument, you cannot specify a remote file-server host in the path name of the file database. The `ostest` utility passes the operation to a local native utility. If you specify a remote file-server host name, ObjectStore informs you that you specified an illegal path name.

The `ostest` utility cannot be used on Windows to test for a file database.

API

See `os_dbutil::stat()` on page 191 in the *C++ API Reference*.

osverifydb

The `osverifydb` utility verifies all pointers and references in a database including internally managed ObjectStore segments. Note that, as used here, the term *references* refers to soft pointers and ObjectStore references.

Syntax

```
osverifydb [options] pathname
```

Argument

pathname

Specifies a file or rawfs database whose pointers you want to verify.

Options

`-end_cluster` *`cluster_number`*

When used with the `-start_cluster` option, this option specifies the range of clusters to verify within a segment. This option must also be used with the `-segment_n` option.

`-end_offset` *`integer`*

When used with the `-start_offset` option, this option specifies the range of offset addresses (in bytes) to verify within a cluster. If this option is not specified, the default is to verify up to the end of the cluster.

The `-end_offset` option must also be used with the `-segment_n` option and with the `-start_cluster` and `-end_cluster` options. Also, the values specified by the `-start_cluster` and `-end_cluster` options must equal the same cluster number.

`-F`

The `-F` option (uppercase) enables `osverifydb` to operate in "fast mode".

The `-F` option cannot be be coupled with the following `osverifydb` options:

- `-ignore_references`
- `-illegal_pointer_action`
- `-o`
- `-v`
- `-whohas`

`-ignore_references`

Suppresses verification of references. For more information about using this option, see Verifying references on page 260.

`-illegal_pointer_action { null | ask }`

If the `-illegal_pointer_action` option is used with the `null` argument, `osverifydb` sets illegal pointers to null. If used with the `ask` argument, `osverifydb` prompts the user for a reference value. If the user responds to the prompt by pressing the Return key, `osverifydb` skips this illegal pointer. If you do not specify the `-illegal_pointer_action` option, `osverifydb` flags the illegal pointer as an error without taking any action.

`-L transaction_log_name`

For use with ObjectStore / Single only. When this option is specified, the named file is used for the transaction log file. If a file with the name *transaction_log_name* already exists, it must be a properly formed transaction log. By default, `osverifydb` uses a temporary file.

`-nocoll`

Suppresses integrity checks that ensure that the ObjectStore collections in the database are valid. ObjectStore Technical Support recommends that you use this option only on databases that do not contain collections.

`-no_internal_segments`

Skips verifying internally managed segments.

`-o`

Displays each object in the database using the Metaobject Protocol (MOP).

`-segment_n segment_number`

Verifies only the segment specified by *segment_number*. By default, `osverifydb` verifies all segments in the database, including internal segment `0`.

`-start_cluster cluster_number`

When used with the `-end_cluster` option, this option specifies the range of clusters to verify within a segment. This option must also be used with the `-segment_n` option.

`-start_offset integer`

When used with the `-end_offset` option, this option specifies the range of offset addresses (in bytes) to verify within a cluster. If this option is not specified, the default is to start verifying at the beginning of the cluster.

This option must also be used with the `-segment_n` option and with the `-start_cluster` and `-end_cluster` options. Also, the values specified by the `-start_cluster` and `-end_cluster` options must equal the same cluster number.

`-v`

Displays the value for each pointer.

`-whohas pointer`

Lists objects that point to the object indicated by *pointer*. The *pointer* argument can take the form either of an ObjectStore reference in dump format or of the following:

segment`:`*cluster*`:`*offset*

where *segment*, *cluster*, and *offset* specify the object's segment number, cluster number, and offset into the cluster.

Values can be in decimal or hexadecimal format. Hexadecimal values must be prefixed by the characters `0x`.

Exit codes

The `osverifydb` utility returns `0` if no errors were found; otherwise, it returns a nonzero value.

Description

Verification means this:

- There are no transient pointers.
- Persistent pointers point to valid (not deleted) storage.
- The declared type for a pointer as determined from the schema matches the actual type of the object pointed to.
- References are consistent.

When `osverifydb` detects an invalid pointer, it indicates the location and the value of the pointer. Whenever possible, it displays a symbolic path to the bad pointer, starting with the outermost enclosing object.

The `osverifydb` utility runs integrity checks to ensure that the ObjectStore collections in the database are valid. You can suppress verification of collections by specifying the `-nocoll` option when you run `osverifydb`.

Verifying references

Before a reference can be verified, must be resolved to an address. The process of resolving addresses consumes address space, and, in some cases, may cause the `osverifydb` utility to run out of address space. If the utility fails because it requires more address space than is available, you should use the `OS_AS_SIZE` environment variable; for more information, see `OS_AS_SIZE`.

As a last resort, you can also use the `-ignore_references` option to turn off reference verification. Note that you should not normally use this option unless you have already tried to verify the database and verification failed because `osverifydb` ran out of address space.

How often

How often you should verify database pointers and references depends on how often your data changes. Verifying databases before backups is a good practice, but verification can be time consuming. You might want to verify databases every evening.

Verifying indexes and index entries

If the `OS_COLL_DEBUG_INDEX` environment variable is set, `osverifydb` will check the consistency of the type of all indexes on each collection with the type of each member of that collection. For more information, see `OS_COLL_DEBUG_INDEX`.

Fast mode

The fast mode for `osverifydb` uses techniques optimized for performance purposes. Run `osverifydb` with the `-F` option to specify fast mode. If you set the environment variable `OS_OSVERIFYDB_FAST` to true (non-zero), `osverifydb` will run in fast mode by default.

You can use the environment variable `OS_OSVERIFYDB_BUFFER_SIZE` to specify the maximum size of the transient buffer used to optimize locality of reference by `osverifydb` in fast mode. The default size is 512 MB.

The fast mode for `osverifydb` is not compatible with the following options:

- `-ignore_references`

- -illegal_pointer_action
- -o
- -v
- -whohas

Schema protection

If you are developing an application and will need to run this utility on a protected schema database, ensure that the correct key is specified for the environment variables OS_SCHEMA_KEY_LOW and OS_SCHEMA_KEY_HIGH. If the correct key is not specified for these variables, the utility fails. ObjectStore signals

```
err_schema_key _CT_invalid_schema_key,
"<err-0025-0151> The schema is protected and the key provided did not
match the one in the schema."
```

API

See os_dbutil::osverifydb() on page 186 and os_dbutil::osverifydb_one_segment() on page 187 in the *C++ API Reference*.

Examples

```
osverifydb -illegal_pointer_action null vtest1.db
```

The null argument causes osverifydb to null all illegal pointers.

```
osverifydb -illegal_pointer_action ask vtest2.db
```

The ask argument permits selective repair; that is, it causes osverifydb to prompt for an alternative value for the illegal pointer. Following is sample output from osverifydb in such a circumstance:

```
The object at
<d|red:E:\test\test.db|2|0|545000|10000>(type "SomeColor"),
contains a pointer at <d|red:E:\test\test.db|2|0|545194|10000>
()My_AlarmMessage.m_pValue) incorrectly pointing yo
<d|red:E:\test\test.db|2|0|57afc0|10000>
```

Bad pointer error format

osverifydb outputs information about bad pointer errors in the following format:

```
< format | db_path | seg_num | obj_info | hex_offset | size >
```

where the different fields have the following meanings:

- *format* can be the letter e or d. An e specifies that the *obj_info* field is an export ID; d specifies that the *obj_info* field is a cluster number.
- *db_path* is the absolute path name of a database.
- *seg_num* is a segment number.
- *obj_info* is either an export ID or a cluster number (see *format*, above).
- *hex_offset* is a byte offset in hexadecimal.
- *size* is optional. If present, it represents the mapping granularity in hexadecimal — typically 10000, corresponding to a size of 64 KB.

You can then press Enter, in which case the illegal pointer is set to null, or you can enter a valid reference string such as /daffy/home/daffy/daffy0/dbs/verifydb1 | 2 | 64 identifying an object at offset 64 in segment 2, in the database verifydb1. The new pointer value, if valid, is used as the replacement value for the pointer in the database.

Caution

It is very important to use the `null` option with caution; using it indiscriminately can result in a corrupted database.

The following output is the result of running `osverifydb` on a database that contains an object of type `c1`, with the bad pointers identified by the error messages.

```
beethoven% osverifydb /camper/van

Verifying database beethoven::/camper/van
Verifying segment 2 Size: 8192 bytes

Pointer to nonpersistent storage.
Pointer Location: 0x6010000. Contents: 0x1.
Lvalue expression for pointer: c1::m1

Pointer type mismatch; the declared type is incompatible with the
actual type of the object
Pointer Location: 0x6010004. Contents: 0x601003c.
Declared type c2*. Actual type: c3*.
Lvalue expression for pointer: c1::m2

Pointer to deleted storage
Pointer Location: 0x6010008. Contents: 0x6010040.
Declared type c2*.
Lvalue expression for pointer: c1::m3

Pointer type mismatch; the declared type is incompatible with the
actual type of the object
Pointer Location: 0x601000c. Contents: 0x6010028.
Declared type c2*. Actual type: c1*.
Lvalue expression for pointer: c1::m4
Lvalue expression for pointed to object: c1::ma[5]

Pointer type mismatch; the declared type is incompatible with the
actual type of the object
Pointer Location: 0x6010010. Contents: 0x6010044.
Declared type c2*. Actual type: char*.
Lvalue expression for pointer: c1::m5
Lvalue expression for pointed to object: char[0]

Pointer to nonpersistent storage.
Pointer Location: 0x6010068. Contents: 0x1.
Lvalue expression for pointer: void*[5]
Verified 5 objects in segment

Verified 5 objects in database
beethoven%
```

The following is a portion of the output displayed when `osverifydb` is run with the `OS_COLL_DEBUG_INDEX` environment variable set:

```
DBG: Checking indexes on collection at 0xe10b0000
DBG: Index of type "A*"
DBG: Checking with element at 0xe10b08d4 of type "A" OK
DBG: Checking with element at 0xe10b08e8 of type "A2" OK
  ("A" is a base of "A2")
DBG: Checking with element at 0xe10b08d8 of type "A" OK
DBG: Checking with element at 0xe10b08dc of type "B"
  INCONSISTENCY
```

osversion

The `osversion` utility displays the version of ObjectStore that is in use on your machine.

Syntax

```
osversion
```

Description

API

See the following in the *C++ API Reference*:

- `objectstore::release_maintenance()` on page 80
- `objectstore::release_major()` on page 80
- `objectstore::release_minor()` on page 80
- `objectstore::release_name()` on page 81

Also see the ObjectStore header file `include/ostore/osreleas.hh`.

Examples

The first example is for SPARCstation:

```
elvis% osversion
ObjectStore Release 6.1 for Solaris 2.x (SunOS 5.x) Sparc/Sun ONE Studio
7 (C++ 5.4)
```

The next example is for Windows.

```
[D\:] osversion
ObjectStore Release 6.2 for 32-bit Windows Systems, Visual Studio .NET 2002
```

Chapter 5
Using Locator Files to Set Up Server-Remote Databases

This chapter provides information on using a locator file to set up access to databases that do not reside on the same host as a server.

The topics discussed include the following:

What Is a Server-Remote Database?

When an ObjectStore application accesses a file database, the ObjectStore server handling that access is required to be running on the file server host containing the database; that is, the database must be *server-local*. However, you can override this default for file databases and allow access to *server-remote* databases, that is, access to databases not stored on an ObjectStore server host.

Note

This discussion of server-remote databases applies only to file databases and not to rawfs databases.

What Are the Advantages?

The advantages of server-remote databases are as follows:

- Individual ObjectStore users can create databases on a host without first ensuring that an ObjectStore server is running there.
- Databases can be stored on dedicated file servers.

What Are the Disadvantages?

The disadvantages of server-remote databases are that

- Network overhead increases.
- Performance is slower than for server-local databases.
- Network failure can cause a database to be inaccessible or to become inconsistent.

Cautions

The following cautions should be observed:

- You must use NFS to access server-remote databases. When using NFS, you cannot be sure that a write transaction actually completes. This is because NFS is a stateless protocol. (This is a problem when you are modifying any file by means of NFS, not just a database.)
- Using Samba or PC–NFS (or any other Windows-to-UNIX networking protocol) to access a file database stored on a UNIX host from a Windows server is not supported.
- During access of server-remote databases with NFS, if the file host crashes or suffers a network outage, the ObjectStore server is likely to hang until the file host comes back up. This can cause other clients to wait.

How Do You Allow for a Server-Remote Database?

After you set up the appropriate hardware and software to connect your systems, there are two ObjectStore-specific steps for allowing server-remote databases:

- Provide a *locator file*, described next, on the host of the client that needs to access the server-remote database:
- Set the server parameter `Allow Remote Database Access` to `Yes` for each server with the potential to access a database that is not local.

Locator file

ObjectStore clients use locator files to determine the ObjectStore server that should handle access to specific server-remote databases. Locator files need not include information about server-local databases.

Put the locator file in the directory `$OS_ROOTDIR/etc` and name it `locator`. If the file known to a client host as `$OS_ROOTDIR/etc/locator` exists, ObjectStore uses it to determine the server that should handle access to a server-remote database for ObjectStore applications running on that client. It does not matter whether there is a local ObjectStore server. If there is a locator file, ObjectStore uses it.

When you use a locator file, you can use an `$OS_ROOTDIR` directory that is shared by multiple machines on a network.

You do not need a locator file if you do not have server-remote databases.

Overriding locator file

ObjectStore clients can override this specification of a locator file and specify their own locator files with either a client environment variable or a function call. These are discussed in Overriding the Default Locator File on page 277.

Server parameter Allow Remote Database Access

If ObjectStore determines from a locator file that a particular ObjectStore server should handle access to a particular server-remote database, and that server has a value of `Yes` for the parameter `Allow Remote Database Access`, the server handles access to the database. If the server does not have a value of `Yes` for `Allow Remote Database Access`, the exception `err_file_not_local` is signaled.

One server for each database

You should assign only one ObjectStore server to a server-remote database. This server would handle all access to the remote-server database by all applications. This is because one server handling access to a database can prevent concurrent access by other ObjectStore servers. This is discussed further in When Multiple Servers Can Concurrently Access a Database on page 278.

Prototype

When you have many server-remote databases, network overhead increases and performance is slower than for local databases. If you are considering having server-remote databases, it is prudent to set up a prototype and determine whether it meets your needs.

Description of the Locator File

The locator file contains one or more *host declarations* and one or more *locator rules*. It can also include comments.

Each host declaration specifies a host name or a group of host names and the host machine type.

Each locator rule specifies

- The name of the host (file server) in which the database resides
- The path name of a database or group of databases
- The ObjectStore server that controls access to the specified database
- The way to translate the database name from its form on the file server to the form needed by the ObjectStore server

Format

Each locator file has the following format:

```
host-declaration-1
host-declaration-2
...
host-declaration-n
locator-rule-1
END
locator-rule-2
END
...
locator-rule-n
END
```

The format of the locator file also allows it to be used for implementing failover. For more information, see Locator File: FAILOVER_SERVER Declaration on page 291.

Example of a Locator File

The following example file contains two host declarations, one locator rule, and one comment:

```
HOST redwood unix
HOST oak pc

FILE_HOST oak
FILE_PATHNAME c:\oak1\test1\.+
SERVER_HOST redwood
REPLACE c:\oak1\test1 \suite1
REPLACE_DELIMITERS
END

#end of locator file
```

Host declarations

The host declarations are the first two lines. They inform ObjectStore about the operating systems running on the hosts referred to in the locator rules that follow. This allows ObjectStore to execute the `REPLACE_DELIMITERS` command, explained on page 272.

Locator rule

The locator rule in this file indicates that the ObjectStore server on the host named `redwood` controls access to any file database that meets both of these conditions:

- The database resides on the file server host named `oak`.
- The path name by which it is known to that file server is `c:\oak1\test1\` followed by one or more characters (any characters).

The locator rule specifies that the path name by which this file database is known to the ObjectStore server (`redwood`) can be obtained by

1. Replacing `c:\oak1\test1` with `\suite1` in the path name by which the database is known to the file server
2. Replacing the standard delimiter used by `oak`'s operating system (\) with the standard delimiter used by `redwood`'s operating system (`/`)

The `END` statement indicates the end of a locator rule.

Applying the rule

If an application contains the call

```
os_database::open("/suite1/test1.db");
```

and the host of the application has `oak`'s directory `c:\oak1\test1` mounted as `/suite1`, ObjectStore translates `/suite1/test1.db` into a path name that is canonical for the file server host `oak`; that is, `c:\oak1\test1\test1.db`. This translation is the path name by which the database is known to the file server, and it is used as input to the locator file.

The preceding rule applies in this case because the file server host has the specified name (`oak`) and the input path name `c:\oak1\test1\test1.db` matches the pattern in the rule: `c:\oak1\test1\.+`.

The rule specifies that the ObjectStore server on `redwood` should handle access to the database. The rule also specifies that this database is known to `redwood` as `/suite1/test1.db`, the result of substituting `\suite1` for `c:\oak1\test1` in `c:\oak1\test1\test1.db` and then substituting "`/`" for "`\`".

A database to which the locator rule does not apply is handled as if there were no locator file; it is handled by the ObjectStore server running on the file server host containing the database.

Comments

Comments begin with # and extend to the end of the line.

Case sensitivity

Case is not significant for keywords in locator files.

Declaring Hosts

A locator file begins with one or more host declarations. Each host declaration has the form

```
HOST host-name-pattern {unix | pc}
```

where *host-name-pattern* is a character string pattern that specifies a set of host names. The rules about the way character string patterns are written and used are discussed in Using Character String Patterns in Locator Files on page 273. For example, to specify that all hosts are UNIX systems, you would include this statement:

```
HOST .* UNIX
```

ObjectStore uses the earliest HOST declaration whose *host-name-pattern* matches a given host name. The declaration that ObjectStore uses for a given host name determines the host's associated standard delimiter as used by the REPLACE_DELIMITERS translation command. See Specifying Translation Commands on page 271.

The following is the way a host declaration determines an associated standard delimiter:

- For the UNIX operating system, the delimiter is / (forward slash).
- For a PC, the delimiter is\ (backslash).

Specifying Locator Rules

After the host declarations, a locator file contains a sequence of one or more locator rules. Each rule has the following form:

Syntax for locator rules

```
FILE_HOST file-server-host-name-pattern
FILE_PATHNAME pathname-pattern
SERVER_HOST ObjectStore-server-host-name
[translation-command-1
 translation-command-2
...
 translation-command-n]
[access-specification]
END
```

When does a rule apply to a database?

ObjectStore applies a locator rule to a database when these conditions are both true:

- The database resides on a file server host whose name has the form specified in the `FILE_HOST` statement.
- The name by which the database is known to the file server has the form specified in the `FILE_PATHNAME` statement.

The first locator rule in a file that applies to a database determines

- The ObjectStore server to handle access to that database
- The path name by which that database is known to that ObjectStore server

If ObjectStore does not find a rule that applies to a database, the server running on the host containing the database handles access to the databases. This always occurs when there is no locator file.

Specifying FILE_HOST Statements

A `FILE_HOST` statement specifies the host name of a system in which a server-remote database is stored. The statement has the form

`FILE_HOST` *`file-server-host-name-pattern`*

where *`file-server-host-name-pattern`* is a character string pattern for the name of a file server. See Using Character String Patterns in Locator Files on page 273.

If a database specified in an application resides on a file server host whose name has this form, the rule containing this statement might apply to the database (see the `FILE_PATHNAME` statement that follows). If the database's file server host has a name that does not match *`file-server-host-name-pattern`*, the rule does not apply to the database.

There is one special *`file-server-host-name-pattern`*:

`@LOCALHOST`

This value indicates the host of the ObjectStore application.

Specifying FILE_PATHNAME Statements

A `FILE_PATHNAME` statement specifies a path name by which a database is known to its host. This statement has the form

`FILE_PATHNAME` *`pathname-pattern`*

where *`pathname-pattern`* is a character string pattern. See Using Character String Patterns in Locator Files on page 273.

If the name by which a database is known to the file server matches *`pathname-pattern`*, the rule containing this statement applies to the database if the database's file server host has a name that matches the rule's *`file-server-host-name-pattern`*. If the database's name does not match the *`pathname-pattern`*, the rule does not apply to the database.

Specifying SERVER_HOST Statements

The SERVER_HOST statement specifies the name of a host running an ObjectStore server. This statement has the form

SERVER_HOST *ObjectStore-server-host-name*

where *ObjectStore-server-host-name* is the name of a host running an ObjectStore server. You can enclose this name in quotation marks (" "). If a locator rule applies to a database, this statement specifies the name of the ObjectStore server that handles access to the database.

There are two special *ObjectStore-server-host-name*s:

- @LOCALHOST specifies the ObjectStore server running on the host of the application.
- @INVALID_SERVER indicates that ObjectStore should signal an error if the locator rule including it applies to a database.

Specifying Translation Commands

Each locator rule can contain an optional sequence of one or more translation commands. If a rule applies to a database, this sequence specifies the way to translate from the name by which a database is known to the file server containing it to the name by which the database is known to the ObjectStore server specified by the rule.

A translation command has one of the following forms:

- REPLACE *pattern substitution-string*
- REPLACE_DELIMITERS
- ALL_UPPERCASE
- ALL_LOWERCASE

Execution order

All REPLACE commands are executed before any other commands in the sequence of translation commands. The first REPLACE command in the sequence applies to the input path name. This is the path name by which a database is known to the file server on which it resides. Each subsequent REPLACE command applies to the output of the previous REPLACE command.

The output of the last REPLACE command is then used as input to the REPLACE_DELIMITERS command (if there is one). The output of that is then used as the input to the ALL_UPPERCASE or ALL_LOWERCASE command (if there is one). Supplying both an ALL_UPPERCASE and an ALL_LOWERCASE command results in an error.

REPLACE syntax

The REPLACE command has the form

REPLACE *pattern substitution-string*

It indicates that the first substring of the input path name (FILE_PATHNAME *pattern*) that matches *pattern* should be replaced by *substitution-string*.

The *substitution-string* can be empty and is optionally enclosed in quotation marks (""). If you want the quotation marks to be interpreted literally, use two consecutive quotation marks, for example ""a_string_with_quotes"".

The special variable `$f` (where `$` is the escape character) designates the string that matches the database host name in the current rule.

Parentheses (`(` and `)`) in *`pattern`* do not affect the pattern that is matched, but they make it possible for a replacement string to include the input substring that matches the parenthesized part of the character string pattern. The variable `$1` in a *`substitution-string`* refers to the substring that matches the first parenthesized expression in *`pattern`*, `$2` refers to the substring that matches the second parenthesized expression, and so on.

You need not specify an escape character for parentheses in character string patterns.

Any string designated by a variable such as `$1` appears as all lowercase in the output.

Example

For example, if the string that matches *`file-server-host-name-pattern`* in the current rule is `inuk` and the input string is

```
/kayak/usr1/foo/bar/file.db
```

then the translation command

```
REPLACE /foo/(bar)/file$.db  /newdirectory/$1/$f/file.db
```

generates

```
/kayak/usr1/newdirectory/bar/inuk/file.db
```

The first two elements of the input string are used unchanged. The replacement begins with the third element, `/foo`, and proceeds as follows:

Original Expression	*Replacement Expression*	*Explanation*
`/(bar)`	`$1`	The `$1` variable matches the first parenthesized expression, in this case, `bar`.
Not applicable	`$f`	The variable `$f` says to substitute the database host name, in this case, `inuk`.
`file$.db`	`file.db`	Substitutes `file.db` for `file$.db`. The `$` in the original expression is required as an escape character for the file extension delimiter, a period (`.`).

REPLACE_DELIMITERS

The `REPLACE_DELIMITERS` command indicates that each occurrence in the input of the standard delimiter associated with the file server host should be replaced with the standard delimiter associated with the ObjectStore server host. See Declaring Hosts on page 269.

ALL_UPPERCASE

The `ALL_UPPERCASE` command indicates that the output should be the same as the input except that lowercase characters in the input should appear in uppercase in the output.

ALL_LOWERCASE

The `ALL_LOWERCASE` command indicates that the output should be the same as the input except that uppercase characters in the input should appear in lowercase in the output.

Specifying Read or Write Access

Each locator rule can contain an optional access specification. This specification is one of

- `READ_ONLY`
- `READ_WRITE`

ObjectStore signals an error if the access specification does not match the access specified in the call to `os_database::open()` or `create()`.

Using Character String Patterns in Locator Files

In some parts of locator files, you can specify a character string pattern. This allows you to write rules that can apply to more than one specific input.

HOST declaration example

For example, if all machines using ObjectStore use a PC syntax for file path names, a single `HOST` declaration that covers them all would be

```
HOST .+ pc
```

The alternative to using a character string pattern is to write a `HOST` declaration for each PC, for example:

```
HOST foo pc
HOST bar pc
HOST amnesiac pc
HOST snoball pc
```

Specifying the character string pattern `.+` is better than listing each host because you need not modify the locator file when you add a new PC host to the network.

Directory specification example

For another example, suppose that there is a directory named `c:\home\place\stuff` that contains many files and directories, including

```
c:\home\place\stuff\more\file.db
c:\home\place\stuff\less\deeper\nota.db
```

You want to describe all files and directories that are in the directory `c:\home\place\stuff` and below. You can specify this with the character string pattern

```
c:\home\place\stuff\.+
```

Maximum length

The maximum length of a character string pattern is 512 characters.

There are three rules for writing character string patterns. An understanding of the following terms is needed to apply the rules correctly.

Definitions of terms

The *target* is character string input. For example, the path name an application specifies for a database is the target.

The *pattern* is a character string pattern in a locator file.

Pattern matching is the process that compares a target with a pattern.

If the target matches the pattern, the result is `true`. If the target does not match the pattern, the result is `false`.

Rules for Writing Character String Patterns

These are the three rules for writing character string patterns.

Pattern and target can be the same

You can specify the pattern as the exact characters that are expected in the target.

This is the simplest kind of pattern. For example, if the only UNIX host is a machine named `dog`, the simplest `HOST` declaration that contains a pattern that matches the target is

```
HOST dog unix
```

When pattern matching seems to be causing problems, it often helps to simplify all patterns according to this rule.

Use meta-characters

You can specify metacharacters in a pattern. This allows a single pattern to match more than one target.

The table in the next section describes the metacharacters you can specify in a pattern. Metacharacters you can specify were chosen to avoid confusion with characters that commonly occur in targets.

A frequently used metacharacter is the period (`.`), which matches any single character. Another is the plus sign (`+`), which matches one or more repetitions of what it immediately follows. For example:

```
.+
```

This matches any target that is at least one character long.

Mix exact strings with meta-characters

You can build complicated patterns from simple patterns by following one with another. For example:

```
dog.+
```

This matches any target that is at least four characters long and whose first three characters are `dog`.

Using Metacharacters in Patterns

The following table describes the metacharacters you can specify in a pattern.

Metacharacter	*Description*
`.`	A period represents any single character. A period matches a single character in the target.
*	An asterisk represents zero or more repetitions of the group to the immediate left. A group is one of the following: • Single character • Period metacharacter • Something enclosed in parentheses • Something enclosed in brackets ([ ]) For example: • `a*` matches `""`, `a`, `aa`, `aaaaaaa`, and so on. • `.*` matches anything, including nothing. • `(abc)*` matches `""`, `abc`, `abcabc`, and so on. • `[05-9]*` matches `""`, `0`, `567`, `98`, and `99999960`, but not `1234,` and so on.
+	A plus sign represents one or more repetitions of the group to the immediate left. A group is one of the following: • Single character • Period metacharacter • Something enclosed in parentheses • Something enclosed in brackets ([ ]) For example: • `a+` matches `a`, `aa`, `aaaaaaa`, and so on. • `.+` matches anything, except nothing. • `(abc)+` matches `abc`, `abcabc`, and so on. • `[05-9]+` matches `98`, `90`, `99999960005`, and so on.

Metacharacter	*Description*
[and]	Brackets enclose a character class. A character class is a special kind of pattern. If any character in the character class matches a single character in the target, the result is true. For example, the character class [abcd] matches the target a, but not e. Inside the brackets, the metacharacters' meanings do not apply, with two exceptions. When a caret (^) is the first character after the left bracket, the sense of the match is reversed. This means that the result is false when a target matches any character in the character class. For example, [^a] matches any single character except a. A caret has no special meaning when it appears in a character class but is not the first character. For example, [a^] matches either a or ^ and nothing else. When a hyphen (-) is not the first or last character in a character class, the result is true for everything that is in the range of the two values on either side of the hyphen. For example, the character class [0-9] matches any single digit. If a hyphen is the first or last character, it has no special meaning. The exact set of characters defined when you specify a hyphen depends on the collating sequence of the machine, so caution is advised. It is safe to assume that [0-9] means the 10 digits on any ASCII system.
(and)	An open parenthesis starts a grouping. A close parenthesis ends a grouping. Parentheses group multiple characters so the software can treat them as a unit. This is useful if sequences might repeat in the target, as shown in the earlier examples for asterisk and plus. Parentheses are also especially useful in delineating a grouping when you are doing string replacements. See the discussion of the REPLACE command in "REPLACE syntax" on page 271.
$	A dollar sign is an escape character. The purpose of the escape character is to provide a workaround in cases in which the target might contain something that is a metacharacter if it appears in a pattern. For example, if the target is +, the pattern + would not match it because the plus sign is a metacharacter and has a special meaning in patterns. You need to escape the plus sign in this way, $+, when you want it to match a target with the value +. You must escape the following characters when they are not used as metacharacters: . * + [ ] ( ) $ ^ . The backslash (\) character has no meaning as a metacharacter; you need not use an escape character with it. The escape character is different from the other metacharacters because you can change it. However, it is not a good idea to do so. By default, the escape character is the dollar sign. You can use the OS_LOCATOR_ESCAPE_CHAR environment variable to change the escape character to something other than the dollar sign. See OS_LOCATOR_ESCAPE_CHARACTER on page 116.
" "	Quotation marks optionally start and end a pattern. For example, "(abc)" is the same as (abc). You can use quotation marks for clarity. To embed quotation marks in a pattern or target, specify two consecutive quotation marks. For example, "a""a" is the three-character pattern made up of lowercase a, quotation marks, and lowercase a.
^	When you specify a caret outside a character class, precede it with the escape character. For example, "$^". Outside a character class, the use of the caret is reserved unless you precede it with an escape character.

Overriding the Default Locator File

The locator file for all ObjectStore clients is

UNIX	`$OS_ROOTDIR/etc/locator`
Windows	`%OS_ROOTDIR%\etc\locator`

To specify a different locator file for a particular client or application, you can either call an ObjectStore function from the application or set a client environment variable.

Calling an ObjectStore Function

The `objectstore::set_locator_file()` function specifies a locator file. Its declaration is

```
static void set_locator_file(const char *file_name)
```

The *`file_name`* argument points to the name of the locator file to be used the next time a database is opened. If you specify `0` for *`file_name`*, the application uses the client environment variable `OS_LOCATOR_FILE` to determine the locator file. A nonzero argument overrides any setting of `OS_LOCATOR_FILE`. If the specified file does not exist, ObjectStore signals the `err_locator_misc` exception. If the first character of the string pointed to by *`file_name`* is a white-space character or `#`, ObjectStore assumes that the string is the contents of a file rather than a file name.

The `objectstore::ignore_locator_file()` function ensures that no locator file is associated with the application, regardless of the setting of `OS_LOCATOR_FILE` or calls to `set_locator_file()`.

Setting a Client Environment Variable

You can set the client environment variable `OS_LOCATOR_FILE` to any legitimate argument for `objectstore::set_locator_file()`, with the same meaning. Calls to `set_locator_file()` override this environment variable.

You can also set `OS_IGNORE_LOCATOR_FILE` to ensure that no locator file is associated with the application. This overrides all other settings and function calls, including `$OS_ROOTDIR/etc/locator`.

When Multiple Servers Can Concurrently Access a Database

When an ObjectStore server handles access to a server-remote database, the server holds a lock on the database as long as the database is open. If the database is open for read-only, the server holds a read lock; if the database is open for read or write, the server holds a write lock.

Read lock

When an ObjectStore server holds a read lock on a database,

- The server allows read access by other servers.
- The server prevents write access by other servers.

Write lock

When an ObjectStore server holds a write lock on a database the following occurs:

- The server prevents other servers from acquiring either a read lock or a write lock on the database.
- The server allows read access by other applications for which it is providing services.

When ObjectStore blocks a server from acquiring a lock, it signals the `err_database_lock_conflict` exception. ObjectStore does not automatically try again to obtain the lock.

Turn off locking

For servers on UNIX machines, you can turn off database-level locking by setting the server parameter `Allow NFS Locks` to `No`.

Caution

You must use extreme caution if you turn off locking. Concurrent database access by different servers can corrupt the database. A mistake in a locator file can cause unintentional concurrent access of this sort.

For servers on Windows machines, database-level locking is always turned on. See `Allow NFS Locks (UNIX Only)` on page 79 for more information about this server parameter.

The recommended mode of operation is as follows:

- Database-level locking is on (the default).
- Each database that is ever opened for read or write has exactly one ObjectStore server assigned to it. This server handles access to the database by all applications at your site. Follow this recommendation to ensure that two different clients do not contact different servers for access to the same database.

Sample Locator Files

The following locator file is used in an environment where an ObjectStore client runs on a Windows machine. The client application is connected to an ObjectStore server that runs on a UNIX machine, `server_1`. The databases are hosted on a second UNIX machine, `server_2`.

```
HOST server_1 unix
HOST server_2 unix
FILE_HOST server_2
SERVER_HOST server_1
FILE_PATHNAME \\server_1\user_1\.*
REPLACE \\server_1\user_1\dbwork\ \disc\users\user_1\dbwork\
REPLACE_DELIMITERS
END
```

NFS-mounted $OS_ROOTDIR with same architectures

The following is an example of a locator file for an NFS-mounted `$OS_ROOTDIR`. The machine called `registry` has `$OS_ROOTDIR` mounted from `towanda`. The machine called `towanda` has its own `$OS_ROOTDIR` and is running an ObjectStore server. Both machines have the same UNIX architecture. `registry:/home/registry/linda` is NFS-mounted onto `towanda` as `/registry_linda`.

In `towanda`'s `$OS_ROOTDIR/etc` directory, the `towanda_server_parameters` file must be modified to have the line

```
Allow Remote Database Access: Yes
```

The locator file is in `towanda`'s `$OS_ROOTDIR/etc` directory. With this locator file, you can build ObjectStore applications on `registry`.

```
HOST towanda unix
HOST registry unix
```

```
FILE_HOST registry
FILE_PATHNAME /home/registry/linda/.+
SERVER_HOST towanda
REPLACE /home/registry/linda /registry_linda
END
```

A locator file for an application that uses three machines

In this scenario, there are three Windows machines. One machine, `bessie`, is the ObjectStore server host. Another machine, `clover`, is the file server (NT) and is not running an ObjectStore server. The remaining machine, `jeep`, contains the ObjectStore client application.

Following is the locator file that is placed in the `%OS_ROOTDIR%\ETC` directory of the ObjectStore client:

```
HOST bessie pc
HOST clover pc
FILE_HOST clover
FILE_PATHNAME \\clover\share\\my-dir.+
SERVER_HOST bessie
REPLACE \\clover\share\my-dir t:\my-dir
END
```

The ObjectStore server host `bessie` has the file server partition mounted as drive `t:`. This is necessary. You cannot expect the ObjectStore server host to use UNC to access

the file server. The file server must be mounted using the File Manager/Explorer or the `net use` command.

With this locator file, the ObjectStore client could access the database `\\clover\share\my-dir\metaschm.db` in either of the following ways:

- `\\clover\share\my-dir\metaschm.db`
- `bessie:t:\share\metaschm.db`

If the ObjectStore client has `\\clover\share` mounted to itself as drive `G:`, it could also use the following access patterns:

- `g:\my-dir\metaschm.db`
- `metaschm.db`

The last pattern works, assuming that the working directory is `g:\my-dir`.

Mounting $OS_ROOTDIR and mixing UNIX architectures

The following example is a locator file for different UNIX architectures in which one of the machines mounts `$OS_ROOTDIR` from another machine. The machine named `screamer` is a machine with its own `$OS_ROOTDIR` but without a local server. The machine `towanda` is a Sun machine with its own `$OS_ROOTDIR` and a server.

`screamer:/home/screamer/box/linda` is NFS-mounted onto `towanda` as `/screamer_linda`.

Again, in `towanda`'s `$OS_ROOTDIR/etc` directory, the `towanda_server_parameters` file must be modified to have the line

```
Allow Remote Database Access: Yes
```

The locator file is in `screamer`'s `$OS_ROOTDIR/etc` directory. With this locator file, you can build ObjectStore applications on `screamer`.

```
#Good locator file:
HOST towanda unix
HOST screamer unix

FILE_HOST screamer
FILE_PATHNAME /home/screamer/box/linda/.+
SERVER_HOST towanda
REPLACE /home/screamer/box/linda /screamer_linda
END
```

Accessing databases on other server hosts

The next locator file shows the way an application that is local to an ObjectStore server can access databases on other server hosts. The application starts on server host `venus` and then creates (opens) databases on disks that are local to the server hosts `mars` and `pluto`.

```
HOST venus unix
HOST mars unix
HOST pluto unix

FILE_HOST mars
FILE_PATHNAME /local/directory/on/mars/.+
SERVER_HOST venus
REPLACE /local/directory/on/mars /mounted/directory/on/venus
END

FILE_HOST pluto
```

```
FILE_PATHNAME /local/directory/on/pluto/.+
SERVER_HOST venus
REPLACE /local/directory/on/pluto /mounted/directory/on/venus
END
```

Always using same ObjectStore server

The following locator file always forces the use of the same ObjectStore server, regardless of where the database is. In the following example, the hosts are all UNIX systems and `tokyo` is the host of the server that you want all databases to use:

```
HOST .+ unix
FILE_HOST .+
FILE_PATHNAME .+
SERVER_HOST tokyo
END
```

Incorrect REPLACE line in locator file

Following is an example of an incorrect locator file that tries to accomplish the same setup:

```
#Bad locator file: do NOT put machine: in the REPLACE line
HOST towanda unix
HOST screamer unix

FILE_HOST screamer
FILE_PATHNAME /home/screamer/box/linda/.+
SERVER_HOST towanda
REPLACE screamer:/home/screamer/box/linda /screamer_linda
END
#end of locator file
```

This locator file would display a message such as this:

```
schema2.cc line 20: error: the compilation schema database could not
be opened because:   The directory was not found
re /home/screamer/box/linda/tutorial_IIA.comp_schema while
creating file database
/home/screamer/box/linda/tutorial_IIA.comp_schema (host "towanda")
```

Missing matching pattern in locator file

Another example of an incorrect locator file is this:

```
#Bad locator file: Need the "/.+" after the file pathname for matching
HOST towanda unix
HOST screamer unix
FILE_HOST screamer

FILE_PATHNAME /home/screamer/box/linda
SERVER_HOST towanda
REPLACE /home/screamer/box/linda /screamer_linda
END
```

This locator file would display a message such as this:

```
"schema2.cc", line 20: error: the compilation schema database could
not be opened because:
The server refused the connection
Attempted to connect to server on local host, looking up database
"/home/screamer/box/linda/tutorial_IIA.comp_schema" on host
"screamer" failed: <err-0003-0004>Host refused connection.(err_net_
connection_refused). Possibly there is no server running on this host.
You can try the command
"/home/screamer/box/ostore/hp310/bin/ossvrping screamer" to check.
(err_server_refused_connection)
```

NFS Limitations

The ObjectStore server needs to act on behalf of its client processes and so it accesses files over the network, thereby relying on NFS. Where NFS fails, the ObjectStore server also fails. The following table describes NFS problems that can occur when you use locator files. These limitations are the result of known NFS problems and do not necessarily apply when you are using other remote-file-access protocols.

Symptom	*Reason*	*Workaround*
Server crash. The `oss_out` file contains references to stale file handles. This most often occurs in conjunction with using the automounter.	The ObjectStore server needs to access the file handle. When the file handle goes stale, the server loses its ability to communicate with the database, and thus crashes. When the ObjectStore server accesses the database, it does not exercise the automounter. Therefore, when you use the automounter, there is an increased chance of seeing this problem. ObjectStore Technical Support recommends not using the automounter. However, even hard mount points can become stale.	None. ObjectStore Technical Support recommends that you do not use the automounter.
Server crash. The `oss_out` file contains references to permission problems.	NFS is poorly configured. You must configure NFS such that `anon=0`. When an application creates or opens a database, the server acts on behalf of the user requesting the open or create. However, after the database is open, all reads and writes occur as `root`. When `root` attempts to log in to another machine, if `anon=0` is not set, the `root` must log in as the equivalent of `noone`, with no privileges. Setting `anon=0` allows `root` on one machine (the server machine) to have the same privileges as `root` on the other machine (on which the database resides).	Configure NFS with `anon=0`.
Server crash. The `oss_out` file contains references to missing files.	Suppose a server has started accessing a database and someone removes or renames the file. If the removal is done on a machine other than the database host, NFS maintains a copy of the database so the server can continue to access it. However, if the removal occurs on the machine with the database (and the machine is not the server host), NFS does not retain a copy of the database for the server process to continue using. Suddenly there is no database to access, even though the server was already accessing it. It is also impossible to determine where the server was in the access stage.	None.
Server crash. `oss_out` file contains references to remote file system's being full.	There is a Sun bug that prevents state flags from being correctly reset. Consequently, it is possible to fill a file system, remove a file or decrease the size of a file, and have a system flag that continues to report that the file system is full.	Obtain a patch from Sun.

Symptom	*Reason*	*Workaround*
Server hang (wait state).	The server requires the lock manager to be running on the same machine as the server. If the lock manager is not running, the server cannot do its job. The server waits (appears to be hung) for the lock manager to be restarted. After the lock manager is running, the server can and will respond.	When a system reboots, ensure that the lock manager is running before the autostart of the server.
Server hang (wait state).	The response from NFS is slow. If, for any reason, NFS is slow to respond to requests for service, the server must wait for those responses, just as any other client of NFS would.	None.

Troubleshooting

If your locator file is not functioning the way you intend, first check the locator file carefully for the following common mistakes:

- Did you reverse the `SERVER_HOST` and `FILE_HOST` arguments? The `SERVER_HOST` statement specifies the name of the host of the ObjectStore server. The `FILE_HOST` statement specifies the name of the host on which the database you want to access is stored.
- Can the value specified for `FILE_PATHNAME` match the database name provided? If you are only using one directory for a database, you can specify `.+` to match anything. If you are using more than one directory, you can specify something like `/dir/subdir/.+`.
- Did you specify `REPLACE_DELIMITERS` for translation between systems that use different delimiters?
- Did you specify any `REPLACE` statements accurately?

In addition to carefully proofreading the locator file, you can set the `OS_DEBUG_LOCATOR_FILE` client environment variable to `1`. When this variable is set, ObjectStore sends diagnostic information on the processing of the locator file translation to `stderr`.

Chapter 6
High Availability of Data

ObjectStore provides a number of facilities for increased levels of reliability. In descending order of reliability, the tools and capabilities are

- Failover
- Asynchronous replication
- `osbackup` (see General Backup Practices on page 50)
- `osarchiv` (see Archive Logging on page 51)

This chapter discusses the following topics:

Note

ObjectStore's support for failover allows you to perform rolling upgrades. That is, you can perform planned upgrades without having to take the system out of service. For more information, see the installation guide that accompanies the ObjectStore upgrade package.

Failover

The purpose of failover is to ensure that the failure of an ObjectStore server, or the failure of the machine on which the server is running, does not prevent service to ObjectStore clients. When failover is in effect, the failure of the protected server is immediately detected, and its service is picked up by another server running on its own machine, providing clients with continued access to the database.

Hot spare hardware

Failover automatically detects hardware or software failure and resumes service by means of a new `osserver` process running on hot spare hardware. *Hot spare* hardware is hardware that is already installed, powered, and working, and thus is instantly available for service when the primary hardware is unavailable.

Transactions

Transactions that have already committed are recovered from the transaction log. Transactions in progress at the time of the failure are aborted and must be retried. If the client application uses lexical transactions, it does not need to be modified to take advantage of failover. The retry mechanism that is built into lexical transactions ensures that they will be automatically retried. The only change that must be made to the client application code is to ensure that it operates correctly when a transaction is retried. Persistent side-effects are rolled back when the transaction is aborted in

preparation for retry, but transient side-effects are only rolled back if the client application explicitly arranges for that. Use the `basic_undo` class or avoid transient side-effects inside a transaction. For information about the `basic_undo` class, see `basic_undo` in Chapter 2 of *C++ API Reference*.

After a failover occurs, it is important for the administrator to fix the broken primary machine as fast as possible in order to return to a redundant-operation state; otherwise, the secondary machine might fail with nothing standing by to take over for it.

Media failure

Failover is not intended to address storage media failures — only failures in the access path to storage media, such as a SCSI bus. To provide high availability in the face of storage media failures, databases and transaction logs must be stored on a disk system with built-in failure avoidance capability — for example, a RAID-1 (disk mirroring) system. For large, mostly read-only databases, RAID-5 could also be used, but RAID-5 can be slow for writes.

ObjectStore supports two types of failover:

- Built-in failover that is included with the ObjectStore product. This type of failover is referred to in this document as *ObjectStore-managed failover*.
- Failover that is provided by cluster operating systems — referred to as *cluster-managed failover*.

Both types of failover are described in the following sections.

ObjectStore-Managed Failover

ObjectStore provides and manages its own built-in failover system. The following sections describe two different ways to configure ObjectStore-managed failover. For information about enabling ObjectStore-managed failover, see Implementing Failover on page 288.

Two osserver Processes, One Logical Server

In this configuration, the failover system consists of two `osserver` processes, each running on a separate machine. But the system appears as one logical ObjectStore server, hosting databases that are stored on a shared disk system. One `osserver` process is the primary server, which is in active use during normal operations. The other `osserver` process is a secondary server, which is inactive except for monitoring the state of the primary server through ObjectStore's failover heartbeat mechanism. If the primary server fails, the secondary server takes over. For more information about the heartbeat mechanism, see Heartbeat Mechanism (ObjectStore-Managed Failover Only) on page 289.

The two machines are connected to a shared disk system. The transaction log is also stored on the shared disk system. The databases and transaction log can be stored in rawfs or (if the operating system supports a file system that is shared between the two machines) in files. For best performance, the transaction log should be on a separate disk. Note, however, that this is not possible when the transaction log is stored in rawfs.

A FAILOVER_SERVER declaration in the locator file tells ObjectStore clients about the two osserver processes, specifically the host names through which they can be accessed. The declaration also tells ObjectStore clients about the fact that if the primary server fails, the secondary server takes over automatically. The declaration also tells ObjectStore clients how long to retry before timing out and reporting an error. For more information, see Locator File: FAILOVER_SERVER Declaration on page 291.

ObjectStore clients can be run on one or more separate machines, or they can be run on the same pair of machines as the osserver processes if the clients have their own failover mechanism.

Four osserver Processes, Two Logical Servers

This configuration consists of four osserver processes that appear as two logical ObjectStore servers, hosting two sets of databases managed by two transaction logs. The processes are organized into two failover server pairs sharing two machines. A pair consists of a primary server process, which is in active use during normal operations, and a secondary server process, which is on standby.

When both machines are working normally, each machine has one primary server and one secondary server. The secondary server on each machine monitors the state of the primary server in the pair running on the other machine, using ObjectStore's failover heartbeat mechanism; see Heartbeat Mechanism (ObjectStore-Managed Failover Only) on page 289. If a primary server fails, then the secondary server running on the other machine takes over. During a failover condition, both servers that are running on the nonfailing machine are primary, and no secondary servers are running.

In order to use this configuration, the user must partition the dataset into two disjoint sets of databases, with the aim of equalizing the load on the two logical ObjectStore servers. Logical server A serves half the databases and consists of a primary server on machine 1 and a secondary server on machine 2. Logical server B serves the other half of the databases and consists of a primary server on machine 2 and a secondary server on machine 1.

When both machines are working normally, the primary servers are active. Databases on logical server A (those whose pathname starts with A: or A::) are served by machine 1; databases on logical server B (those whose pathname starts with B: or B::) are served by machine 2. If one machine fails, the secondary server on the other machine becomes active and the one working machine will bear the entire load, running one osserver process for logical server A and a second osserver process for logical server B.

The primary servers of the two failover pairs should be on separate machines. To ensure this, the user should not start both servers in a failover pair at exactly the same time. The primary server should be started first and given a lead of at least 3 heartbeat intervals to allow it to become primary, before starting the secondary server. For more information about the heartbeat interval, see Heartbeat Mechanism (ObjectStore-Managed Failover Only) on page 289.

If both primary servers are running on the same machine and you want to move one of them over to a second machine, see the procedure for performing manual failback in Problems When Starting Failover Servers on page 295.

For information about configuring ObjectStore-managed failover as two logical servers, see Configuring More Than One Logical Server on page 293.

Cluster-Managed Failover

ObjectStore is able to work with the Sun Clusters 3.0 operating system. This operating system (hereinafter referred to as the *cluster operating system*) can detect and recover from server failure without having to use ObjectStore-managed failover. However, cluster-managed failover does require that the ObjectStore client be able to recover and retry when its connection to a server is broken.

Cluster-managed failover consists of a cluster of two or more machines and one or more shared disk systems, all managed by a cluster operating system. The cluster operating system monitors the status of `osserver` (and other processes). If a monitored process fails, the cluster operating system restarts it, after substituting good hardware for any failed hardware. The cluster operating system also allows file sharing across the cluster without the reliability problems of NFS.

The clustered system appears as one or more logical ObjectStore servers, each hosting its own set of databases and having its own transaction log. Each logical ObjectStore server has its own network address which clients use to contact it. For information about configuring more than one logical server, see Configuring More Than One Logical Server on page 293.

ObjectStore's failover heartbeat mechanism is not used, and there are no standby osserver processes. There is one `osserver` process for each logical ObjectStore server. If that process or the machine on which it runs fails, the cluster operating system starts a new `osserver` process with the same network address and disks, either on the same machine or on a substitute machine.

After the new `osserver` process is started on a substitute machine and the failed machine is ready to return to service, the cluster operating system kills the `osserver` process on the substitute machine and starts a new process on the original machine. This activity is called *failback*.

Clients can run on the clustered system or on another system. A client references an `osserver` process by its logical host name. (When installing a cluster, you make up a *logical host name* for each resource, such as an osserver process.) If there is more than one `osserver` process running on the cluster, each will have its own logical host name.

The next section describes how to set up cluster-managed failover.

Implementing Failover

You can implement failover either by editing the server parameters file and the locator file as described in the following sections or by running the `osconfig` utility. The `osconfig` utility will report on standard output any changes that the user must

do — for example, copy files to another machine when there is no NFS network connection to allow `osconfig` to perform the copy. For more information about the `osconfig` utility, see osconfig on page 164. Note that `osconfig` cannot be used on Windows machines.

The following sections describe what you must do to implement support for failover. Unless explicitly stated otherwise, the information in these sections applies to both ObjectStore-managed failover and cluster-managed failover.

Heartbeat Mechanism (ObjectStore-Managed Failover Only)

If you are using ObjectStore-managed failover, the parameter file must contain information about the heartbeat. This information is used by ObjectStore to set up the heartbeat mechanism so that a secondary server can monitor the state of the primary server in the failover server pair. Note that the heartbeat mechanism is not used by cluster-managed failover, which provides its own means for monitoring the state of failover servers.

Failover Heartbeat Time parameter

The `Failover Heartbeat Time` parameter specifies how often a heartbeat for the primary server is written to disk and how long it takes the secondary server to recognize that the primary server has failed (five times the heartbeat value). This parameter can be set to between 2 and 60 seconds. For more information, see Failover Heartbeat Time on page 90.

Failover Heartbeat File parameter

ObjectStore can store the heartbeat information in a rawfs partition or in a shared operating-system file. If it is stored in an operating-system file (that is, there is no rawfs or rawfs partition 0 is file-based), the parameter file must include the `Failover Heartbeat File` server parameter. The value specified for this parameter is the pathname of the file where you want ObjectStore to store the heartbeat. ObjectStore automatically creates this file when the rawfs is initialized, even if there is no actual rawfs partition. For more information about this parameter, see Failover Heartbeat File on page 89.

Failover Scripts (ObjectStore-Managed Failover Only)

Failover Script parameter

If you want ObjectStore to execute a script or program when failover occurs and the secondary server takes over from the primary server, you must set the `Failover Script` server parameter, specifying the path of the script or program. For more information about this parameter, see Failover Script on page 90.

This parameter is not required. It is applicable only when implementing ObjectStore-managed failover.

Identifying Shared File Systems

Identical Pathnames on Failover Server parameter

If databases or the transaction log are stored in files (rather than rawfs), or if the rawfs is file-based rather than hard-disk-partition-based, you must specify the `Identical Pathnames on Failover Server` server parameter in the parameter file to inform the server of the configuration. This server parameter tells the server where the shared file system or systems exist in pathname space. If you are using cluster-managed failover, `Identical Pathnames on Failover Server` is the only server parameter you must specify in the parameter file.

For more information about this server parameter, see Identical Pathnames on Failover Server on page 90.

Restarting the Server (ObjectStore-Managed Failover Only)

After setting the failover-specific server parameters, you must restart the ObjectStore server with the `-upgradeRAWFS` option, as follows:

```
osserver -upgradeRAWFS
```

This option causes the server to add the heartbeat either to the file specified by the `Failover Heartbeat File` parameter or to a rawfs partition.

Resource Properties (Cluster-Managed Failover Only)

The following ObjectStore-specific resource properties are relevant to configuring a Sun Clusters 3.0 system that is running an ObjectStore server with cluster-managed failover:

`OS_ROOTDIR : string`

Default: `/global/opt/ODI/OS6.1.0/ostore`

Specifies where ObjectStore was installed. If the value does not specify a global file system path, it must be a node-local path in which ObjectStore has been installed on every node on Nodelist. The value of `OS_ROOTDIR` must be the same on every node.

`osserver_command : string`

Default: `$OS_ROOTDIR/lib/osserver -hostname $HOSTNAME -p` *`file`*

Specifies the command line to start the `osserver` process. The command line cannot contain I/O redirection, `$` substitution of variables other than `OS_ROOTDIR` and `HOSTNAME`, or other complex syntax. `$OS_ROOTDIR` is substituted from the `OS_ROOTDIR` resource property. `$HOSTNAME` is substituted from the logical host name that was specified with `register_osserver -h`.

The `-p` option specifies a location in the global file system for the server parameter file. Specifying this option ensures that the same server parameter file is used no matter which node is currently running the `osserver` process. For information about `-p` and other `osserver` options, see osserver on page 217.

This property is changed when setting up more than one logical server. The user can also change it if necessary.

`Environment : string`

Default: <*`empty`*>

Specifies a `stringarray` of *`var=val`* environment settings for `osserver`. The monitor process also sees these environment settings. Use this resource property for any customization you need to do. You do not need to set the `OS_ROOTDIR`, `PATH`, and `LD_LIBRARY_PATH` environment variables here.

This property is changed when setting up more than one logical server. The user can also change it if necessary.

Locator File: FAILOVER_SERVER Declaration

You must include a FAILOVER_SERVER declaration in the locator file. This declaration tells ObjectStore clients about the failover server so that, if the server fails, the client will try to connect with (in the case of ObjectStore-managed failover) the secondary server or (in the case of cluster-managed failover) the restarted server. It also tells the client how long to retry before giving up and reporting an error.

The FAILOVER_SERVER declaration uses the logical names. Client application programs use the FAILOVER_SERVER logical names, not the ALTERNATIVE_SERVER logical names and not the physical machine names.

Syntax

To declare a failover server, you must include the FAILOVER_SERVER declaration in the locator file that specifies information about the server. The FAILOVER_SERVER declaration has the following syntax:

```
FAILOVER_SERVER logical_server_name_1
  ALTERNATIVE_SERVER logical_server_name_2
  RECONNECT_TIMEOUT integer # in seconds
  RECONNECT_RETRY_INTERVAL integer # in seconds
END
```

Arguments

When you are using ObjectStore-managed failover, the *logical_server_name_1* argument is the primary server in the failover server pair, and the *logical_server_name_2* argument specifies the secondary server of a failover server pair. Note that *logical_server_name_1* is the logical server host name that is recorded in all database path names to databases of this failover server. Both arguments are required.

When you are using cluster-managed failover, the *logical_server_name_1* and *logical_server_name_2* arguments must both refer to the same logical server: the logical host name of the osserver process that you registered with the cluster operating system.

RECONNECT_TIMEOUT specifies the maximum number of seconds that a client can attempt to reconnect to a failover server before signaling the exception err_broken_failover_server_connection.

RECONNECT_RETRY_INTERVAL specifies how often (in seconds) the two failover servers are pinged during the RECONNECT_TIMEOUT. The value of RECONNECT_RETRY_INTERVAL should be less than or equal to RECONNECT_TIMEOUT. It is not useful to specify a value for RECONNECT_TIMEOUT that is more than 10 times the value of RECONNECT_RETRY_INTERVAL. Also, RECONNECT_RETRY_INTERVAL cannot be 0 if RECONNECT_TIMEOUT is nonzero. The exception err_locator_syntax is signaled if these constraints are violated.

If you are using ObjectStore-managed failover, the RECONNECT_RETRY_INTERVAL argument should be four times the heartbeat interval. If you are using cluster-managed failover, it should be 15 seconds for Sun Clusters 3.0. In either case, RECONNECT_TIMEOUT should be 10 times RECONNECT_RETRY_INTERVAL.

The locator file is read the first time it is needed by the ObjectStore run time. The locator file is not read again by the application unless the objectstore::set_

`locator_file()` function is called. After the call, the locator file is reread the next time the ObjectStore run time uses its contents.

ObjectStore-managed failover

When you are using ObjectStore-managed failover, the locator file provides a way of defining pairs of ObjectStore servers that compose a failover server pair. You add a `FAILOVER_SERVER` declaration to the locator file for each failover server pair. If you have configured ObjectStore to enable clients to connect to more than one server, you must include entries for each primary server that is a member of a failover server pair. The declaration tells ObjectStore clients about the host names through which the two `osserver` processes can be accessed. It also tells the clients that, if the primary server in the pair fails, the secondary server will take over automatically.

Configuring for two logical servers

If you configure ObjectStore-managed failover so that there are two logical servers, you must include two `FAILOVER_SERVER` entries in the locator file, one for each logical ObjectStore server. You must also add four `SERVER` entries to the locator file; each entry maps the logical name of an `osserver` process to network addresses (machine names and port numbers).

Cluster-managed failover

If you are using cluster-managed failover, include the `FAILOVER_SERVER` declaration in the locator file to tell the client that the server supports failover and that it should try to reconnect when the server connection is broken. Note that the same host name must be specified in the *`logical_server_name_1`* and *`logical_server_name_2`* arguments to the `FAILOVER_SERVER` declaration.

Note that the `FAILOVER_SERVER` declaration for cluster managed failover has identical host names for the *`logical_server_name_1`* argument and the *`logical_server_name_2`* argument because the cluster operating system maps the logical ObjectStore server's network address to whatever machine is currently running the `osserver` process.

Shared Disks

Failover depends on a shared, reliable disk (such as RAID 1 or RAID 10 mirrored disks) connected to both the primary and secondary servers. The connection might be either by a dual-ported reliable disk (for Sun machines, for example) or by the disk, with the primary and secondary servers being on the same bus (HP systems, for example).

Cluster-managed failover

When using cluster-managed failover, the shared disks can be managed by the cluster operating system's Global File System, which can accomodate files as well as raw partitions. Also, each `osserver` process should have a direct hardware path to its disk system from the node where that `osserver` is running, without I/O operations having to pass through another node. Each logical ObjectStore server should have its own shared disk group, separate from the disk groups used by other logical ObjectStore servers or by other resources in the cluster. This is the only way to get both active `osserver` processes to have a direct hardware path to disk when they are running on different nodes, because the cluster operating system will only activate one of the two I/O paths to a disk system at a time. ObjectStore will work without such a fully redundant configuration, but performance will be lower.

Restrictions

Failover alone cannot guarantee 100% fault tolerance. For example, failover is ineffective if the shared disk should crash or if the network should fail.

Each of the ObjectStore servers that implements a failover server must have the following on one or more disks that are shared by the two machines on which the servers are running:

- Databases
- Transaction log
- rawfs partitions (if used)
- Failover heartbeat file (if used)

The ObjectStore servers must be running on the same software architecture.

Unless the shared disks are managed by a global file system (but not NFS), they can only contain a rawfs. In this case, file databases cannot be used, and the transaction log and failover heartbeat must reside in the rawfs, not in files.

Locator file restrictions

Note the following restrictions when using the locator file:

- The locator file must be local to the client application so that NFS is not involved in a failover situation.
- Do not add `SERVER` entries to pre-6.1 ObjectStore client locator files. Pre-6.1 clients will fail when trying to read a locator file that contains `SERVER` entries.

Removing Failover (ObjectStore-Managed Failover Only)

To reconfigure ObjectStore without ObjectStore-managed failover, you must remove any failover-specific server parameters from the parameter file — including

- `Failover Heartbeat File`
- `Failover Heartbeat Time`
- `Failover Script`
- `Identical Pathnames on Failover Server`

Also, if you configured ObjectStore-managed failover to store the heartbeat in a rawfs partition, you must remove the heartbeat from the rawfs partition by running the server with the `-upgradeRAWFS` option, as follows:

```
osserver -upgradeRAWFS
```

Configuring More Than One Logical Server

ObjectStore enables clients to connect to two different ObjectStore servers running on the same machine. This situation can occur when either ObjectStore-managed or cluster-managed failover is in effect. If a client is connected to different failover servers on different machines and one of the servers fails, then the client can be reconnected to a second server on the nonfailing machine, and both servers can handle requests from the same client at the same time.

The following sections describe entries you must add to the locator file to enable an ObjectStore client to connect to two servers on the same machine at the same time. The first section describes the FILE_HOST declaration, which you must include in the locator file if you are using ObjectStore- or cluster-managed failover. The second section describes the SERVER declaration, which you must also add to the locator file only when using ObjectStore-managed failover.

Parameter File

To run two logical ObjectStore servers on the same machine requires two sets of server parameters. Also, the command line for starting the osserver process must include the -p option to tell it which set of server parameters to use. You must also specify the -server_name or -hostname option so that each osserver process knows its logical host name. For more information about these options for starting an osserver process, see osserver on page 217.

SERVER Declaration (ObjectStore-Managed Failover Only)

The following information about the SERVER declaration applies only if you are using ObjectStore-managed failover.

To enable an ObjectStore client to connect to more than one server on a machine, you must provide information to the client that enables it to access an osserver process by contacting its host machine at one of a set of ports. This information is specified in the locator file, using the SERVER declaration.

SERVER syntax

The SERVER declaration has the following syntax:

```
SERVER server_host_name
  HOST real_host_name
  PORT net portname
  { PORT net portname }
  . . .
  END
```

Arguments

The *server_host_name* argument specifies the name of the logical server that is being mapped to a machine and a set of ports, and the *real_host_name* argument specifies the name of the machine on which the server is running.

The *real_host_name* argument can specify the local machine or another machine on the network.

When choosing a name for a new server or a new machine, the user must ensure that the name is not already in use.

You must provide at least one port specification. Use the *net* and *portname* arguments to the PORT clause to specify a port for server-client communications, as in the following example:

```
PORT IP 51025
```

The string server client:1 is understood in the port specification and is not explicitly written. For additional information about the *net* and *portname* arguments, see Modifying Port Settings on page 59.

By providing more than one port specification, you enable a client to communicate with a server across different networks.

Description

The information specified by the SERVER declaration applies to databases that appear to be on a server named *server_host_name*, either through a *server_host_name*: or *server_host_name*:: prefix in the pathname or through a SERVER_HOST clause in a FILE_HOST declaration later in the locator file; see FILE_HOST declaration on page 295.

Note

All SERVER definitions must appear first in the locator file, before any declarations of failover servers and hosts.

FILE_HOST declaration

If you do not use explicit logical server names as host names in database pathnames (only possible when not using rawfs), you must include FILE_HOST entries in the locator file to tell ObjectStore clients the names of each logical ObjectStore server that hosts each database. There must be separate FILE_HOST entries for each such server.

For more information about FILE_HOST entries, see Specifying Locator Rules on page 269 and Specifying FILE_HOST Statements on page 270. For examples of locator files that include FILE_HOST entries, see Sample Locator Files on page 279.

Environment Variables (ObjectStore-Managed Failover Only)

The OS_PORT_FILE environment variable for each osserver process must be set to point to a port file that specifies unique ports. The SERVER entry in the locator file must use the same ports (see Locator File: FAILOVER_SERVER Declaration on page 291).

Also, the OS_SERVER_OUTPUT_LOG_NAME environment variable for each server should be set to provide the server with a unique text output log file.

For more information about these environment variables, see OS_PORT_FILE on page 120 amd OS_SERVER_OUTPUT_LOG_NAME on page 125.

Problems When Starting Failover Servers

As mentioned in Four osserver Processes, Two Logical Servers on page 287, when starting the servers in a failover pair, the user should delay starting the secondary server (by at least 3 heartbeat intervals) until after the primary server has started. If the two are started at the same time, there is the risk that the primary servers of the two failover pairs will both start running on the same machine, resulting in one machine bearing the entire load. This load-balancing problem can also occur after failover has happened and the failing machine is returned to service.

If both primary servers are running on the same machine, you can move one of the servers over to the other machine by performing manual failback. The ObjectStore clients will see the failback as a failover, and will retry their transactions and continue running. (Note that failback is automatically performed by the cluster operating system, as described in Cluster-Managed Failover on page 288.)

Here are the steps for performing manual failback:

1. Choose which of the two failover pairs is going to experience failback.
2. Use the `ossvrping` utility to make sure that the secondary server for that failover pair is alive on the second machine as a *failover backup server*. For more information about `ossvrping`, see ossvrping on page 245.
3. Kill the primary server for that failover pair on the first machine, using the `ossvrshtd` utility or the UNIX `kill` command. For more information about `ossvrshtd`, see ossvrshtd on page 246.
4. Wait for the secondary server on the second machine to become a *failover primary server*, using `ossvrping` to check its status.
5. Start the server on the first machine. If you used the `osconfig` utility, you should use the startup script that `osconfig` created in the directory you specified. Using this script will ensure that all the correct environment variables and command-line options are provided to `osserver`. For more information about `osconfig`, see osconfig on page 164.
6. Use the `ossvrping` utility to make sure the server you just started on the first machine is alive as a *failover backup server*.

Performance and Independent Disk Systems

For best performance, each logical ObjectStore server should have its own disk system, each set of databases should be on its own set of shared disks, and each transaction log should be on its own shared disk that isn't used for anything else. These disks can be managed as a rawfs or as one or more file system volumes. There can be two rawfs file systems, or (if the operating system supports it) the databases and transaction logs can reside in one or two shared file systems.

The Failover API

Client applications written in C++ can make use of failover features available in the ObjectStore C++ API. Using this API is not required, but it allows applications to access environment information for failover.

The C++ API includes the following functions, most of which belong to the `os_failover_server` class. For complete descriptions of these functions, see the *C++ API Reference*.

- `objectstore::get_locator_file()`
- `os_dbutil::svr_machine()`
- `os_server::get_host_name()`
- `os_server::is_failover()`
- `os_failover_server::get_logical_server_hostname()`
- `os_failover_server::get_online_server_hostname()`
- `os_failover_server::get_reconnect_retry_interval()`
- `os_failover_server::get_reconnect_timeout()`
- `os_failover_server::set_reconnect_timeout_and_interval()`

Exceptions and Error Messages for Failover

This section presents run-time exception cases that can occur with failover.

`err_failover_server_refused_connection`

Signaled when the initial connection to a failover server pair cannot be made.

`err_broken_failover_server_connection`

Signaled when neither the logical nor alternative servers comes back up in some predetermined maximum amount of time that a client process should wait for either server to come up on its own.

`err_server_restarted`

Signaled when a failover server connection is discovered to be lost and then one of the logical or alternative servers comes back up before `RECONNECT_TIMEOUT`. Lexical transactions are restarted when the exception `err_server_restarted` is signaled within them.

Asynchronous Replication

ObjectStore provides the `osreplic` utility, which produces a continuously updated copy (or replica) of one or more user databases. The utility works by coordinating the actions of a source ObjectStore server running archive logger and of a target ObjectStore server running recover, providing a read-only (MVCC) copy of a database that is updated dynamically from the master database.

ObjectStore rawfs databases and rawfs directories can be replicated, as well as ObjectStore file databases. Native file system directories cannot be replicated.

See `osreplic` on page 205 for further information.

Chapter 7
Managing ObjectStore on UNIX

This chapter provides information about managing ObjectStore on UNIX systems. For complete information, you should consult the first six chapters in this book in addition to this chapter.

The topics discussed in this chapter include the following:

Database and Executable Path Names

You specify a file database with an operating system path name. For example:

```
os_database::open("/usr3/fauntleroy/my_file_db")
```

You can also specify a relative path name.

ObjectStore treats links in the usual manner.

Automount path names are acceptable. If you are using `automount`, the path name you specify cannot include the automount prefix, usually `/tmp_mnt`. Referring to the file with this prefix does not cause `automount` to keep the file mounted and can cause the file to appear to be deleted.

Unless you set up a database to be a server-remote database (see Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 265), you must store file databases on a host that is running an ObjectStore server. ObjectStore determines the server to use based on the NFS mountings of the client that creates the database.

Colons in file path names are interpreted as alphabetic characters if a slash character precedes the colon in the path name. For example:

Path Name	*Meaning*
`/usr1/moe/a:b`	Specifies a file named `a:b` in the `/usr1/moe` directory in the local host's name space.
`bill:/usr2/dbs:abc`	Specifies a file named `dbs:abc` in the `/usr2` directory, on server `bill`, in the server's name space.
`fifi/mimi:lulu`	Specifies a file named `mimi:lulu` in the `fifi` directory, relative to the working directory, in the local host's name space.

ObjectStore supports multibyte encoded path names. This includes environments where ObjectStore applications and servers are located on different machines and the client and server machines use different multibyte encodings.

File Name Expansion

When you specify a rawfs database name, ObjectStore commands use wildcard (`*`, `?`, `{}`, and `[]`) name expansion.

You must insert a backslash (\) as an escape character immediately before a wildcard character so that the wild card is expanded by the ObjectStore command and not by the shell, as in the following example `osls` command line:

```
osls sax::/charlie\*
```

Where you specify a file database name, the shell performs wildcard expansion. This means that it does not usually matter whether wildcards are preceded by escape characters (backslashes). For example, the following two commands have an identical effect:

```
osls -d /oshome/bin/osse*
osls -d /oshome/bin/osse\*
```

In the first case, the shell expands the wildcard and the expansions are all passed to the ObjectStore `osls` utility; `osls` determines that they represent file path names and passes them on to the shell command `ls`. In the second case, the shell passes the unexpanded string to `osls`, which determines that it represents a file path name and passes it to the shell command `ls`, which then does the expansion.

Executable Path Names

Path names for the executables for ObjectStore utilities have the following format:

`$OS_ROOTDIR/bin/`*`utility-name`*

ObjectStore utilities and shell commands

ObjectStore utilities have the prefix `os`, and most are analogous to shell commands. The utilities include `oschgrp`, `oschmod`, `oschown`, `osls`, `osmkdir`, `osrm`, and `osrmdir`. In most cases, you can use ObjectStore utilities and their corresponding shell commands interchangeably on operating system files and directories containing ObjectStore file databases. Any differences are noted in the documentation for a particular utility.

Where the input name is a file database name, the commands pass the name on to the corresponding system command. Those commands that modify an individual database file in any way (`oschgrp`, `oschmod`, `oschown`, and `osrm`) attempt to verify that the file being operated on is in fact a database by calling `os_database::lookup()` on the path before operating on it. This verification is only done, however, if neither `-f` (force) nor `-R` (recursive) is specified. If the force or recursive flag is specified, the paths are simply passed to the shell without further checking. This means, for example, that

```
osrm /foo/bar
```

removes `/foo/bar` only if it is a database, while

```
osrm -f /foo/bar
```

removes `/foo/bar` regardless of the type of file that it is.

Note

ObjectStore utilities return a `0` to indicate success and a nonzero value to indicate otherwise.

Setting Server Parameters

Each server parameter that you can specify is described in Chapter 2, Server Parameters, on page 77.

All server parameters have default values. ObjectStore Technical Support recommends that you use the default value for each parameter, which requires no action on your part. You can, however, modify server parameters in two ways:

- You can use the `osconfig` utility; see osconfig on page 164.
- You can create a parameter file, as described in the next section.

After you modify a server parameter, you must shut down and restart the server before any changes to parameters can take effect.

Creating a Parameter File

A parameter file has the following format:

Parameter file format

- One parameter and its value can appear on a line.
- A parameter has a name, such as `Deadlock Victim`.
- The parameter name is followed by a colon.
- The colon is followed by the parameter's value. According to the parameter, this can be numeric or text.
- The colon and value are separated by tabs or spaces.
- Comment lines must contain a # as the first nonblank character.
- Case is not significant for parameter names.
- Embedded spaces are significant in parameter names.
- If the same parameter name appears twice in the parameter file, only the first occurrence is read.

Example

A parameter file can include one or more parameter specifications. The following example provides definitions for the `Log File`, `Partition`*`N`*, and `Authentication Required` parameters:

```
Log File: /support2/local/ostore/sun4.r600/.log
Partition0: FILE/support2/local/ostore/sun4.r600/.part0
Authentication Required: Name Password
```

Search order

When you start an ObjectStore server, ObjectStore looks for a parameter file in the places shown in the following order. It uses the first parameter file that it finds.

1. File specified with the `-p` option to `osserver`
2. `$OS_ROOTDIR/etc/`*`host`*`_server_parameters`, where *`host`* is the value of the `-server_name` or `-hostname` option to `osserver`, or the name of the local host
3. `$OS_ROOTDIR/etc/server_parameters`

`OS_ROOTDIR` is an environment variable set to the top-level directory in the part of the source hierarchy containing ObjectStore files.

Starting the Server

The ObjectStore server usually starts when you boot your system. You can also start the server with the `osserver` utility. Whichever way you start the server, you can configure it with command-line options and parameter file options that the server reads when it starts. For information about the `osserver` utility and its command-line options that are available with the `osserver` utility, see osserver on page 217.

The path name of the server executable is `$OS_ROOTDIR/lib/osserver`. See the installation instructions for setting `OS_ROOTDIR`.

Nonroot Server Startup

Normally, you start the server as `root`. However, this is not required. If you start the server from a non-`root` account, ObjectStore allows access to rawfs databases but does not allow access to file databases. You can change this behavior with the `Restricted File DB Access` server parameter. See Restricted File DB Access (UNIX Only) on page 97.

Creating a Rawfs

Maintaining a rawfs provides a fast, convenient way to manage all ObjectStore databases at your site.

Before you can create a rawfs, you must set aside some partitions to be in the rawfs.

- For raw partitions, you might need to reorganize your disk space unless you have a new disk.
- For a file partition, create a file. If there is anything in the file, ObjectStore overwrites it.

When you create your rawfs, the preferred method of doing so is to use `osconfig rawfs`. This utility prompts you for a path name for the partition and then initializes it. See your installation guide for details.

ObjectStore places no restrictions on the size of a rawfs partition. The rawfs partition size is limited only by the maximum size of a partition allowed by the operating system.

The maximum number of partitions for a single server is 500.

Specifying the Partitions in a Rawfs

The `PartitionN` server parameter specifies the partitions in the rawfs. You add a partition to the rawfs by adding a `PartitionN` statement to the server parameter file. Each `PartitionN` statement has the following form:

`PartitionN: type pathname {expandable | nonexpandable}`

Where

`N`

Specifies a positive integer from 0 to *n*. `PartitionN` statements can appear in the server parameter file in any order, but empty slots are not allowed. For example, if you have four partitions, they must be numbered 0 through 3. Required.

`type`

Specifies whether the partition is a raw partition or a file partition. Required.

For a file partition, the value for *`type`* is `FILE` or `UNIX`, and the value of *`pathname`* is the absolute path name of a UNIX file that is to be used as the *n*th partition in the rawfs.

For a raw partition, the value for *type* is platform-dependent (as described below), and the value of *pathname* must be the raw device name (for example, `/dev/rsd0d`) of a UNIX disk partition to be used as the *n*th partition in the rawfs.

The raw device is also known as a character special file in UNIX. Typically, each partition is accessible both by a raw device and by a block special device. The ObjectStore server requires the use of the raw device.

AIX

On AIX, the value for *type* can be `UNIX`, `FILE`, `PV`, `LV`, or `RAW`:

- `UNIX` or `FILE` indicates the rawfs is stored in an AIX file.
- `LV` indicates the rawfs is stored in an AIX logical volume.
- `RAW` or `PV` indicates the rawfs is stored in an AIX physical volume (raw disk).

HP–UX

On HP–UX, the value for *type* can be `UNIX`, `FILE`, `PARTITION`, or `DISK`:

- `UNIX` or `FILE` indicates that the rawfs is stored in an HP–UX file.
- `PARTITION` indicates that the rawfs is stored in an HP–UX logical volume.
- `DISK` indicates that the rawfs is stored on a raw device.

Linux

On Linux, the value for *type* can be `UNIX`, `FILE`, `PARTITION`, or `DISK`:

- `UNIX` or `FILE` indicates that the rawfs is stored in a Linux file.
- `PARTITION` or `DISK` indicates that the rawfs is stored on a raw device.

SGI IRIX

On SGI IRIX, the value for *type* can be `UNIX`, `FILE`, `LV`, or `PARTITION`:

- `UNIX` or `FILE` indicates that the rawfs is stored in an SGI IRIX file.
- `LV` indicates that the rawfs is stored in an SGI IRIX logical volume.
- `PARTITION` indicates that the rawfs is stored on a raw device.

Solaris

On Solaris, the value for *type* can be `UNIX`, `FILE`, `SLICE`, or `PARTITION`:

- `UNIX` or `FILE` indicates that the rawfs is stored in a Solaris file.
- `PARTITION` or `SLICE` indicates that the rawfs is stored on a raw device or a logical volume.

pathname

Specifies the path of the partition. It must begin with a slash (/). For a raw partition, it is the device name. For a file partition, it is the absolute path name.

AIX

On AIX, specify

- An absolute pathname if you specify `UNIX` or `FILE` for *type*
- The LV name if you specify `LV` for *type*, for example, `oslv`
- The PV name if you specify `RAW` or `PV` for *type*, for example, `hdisk6`

HP–UX

On HP–UX, specify

- An absolute path name if you specify `UNIX` or `FILE` for *type*.
- The logical volume name if you specify `PARTITION` for *type*. You can use the block special device or a raw device name. For example, `/dev/vg00/lvol2` or

`/dev/vg00/rlvol2`, where `00` is the volume group number and `2` is the logical volume.

- The path name for the raw device if you specify `DISK` for *`type`*; for example, `/dev/dsk/c207d6s0`.

Linux

On Linux, specify

- An absolute path name if you specify `UNIX` or `FILE` for *`type`*.
- The path name of the raw device if you specify `PARTITION` or `DISK` as *`type`*; for example, `/dev/rdsk/c1t5d0s2`.

SGI IRIX

On SGI IRIX, specify

- An absolute path name if you specify `UNIX` or `FILE` for *`type`*.
- The path name of the raw device associated with the logical volume if you specify `LV` for *`type`*; for example, `/dev/rdsk/lv0`.
- The path name of the raw device if you specify `PARTITION` as *`type`*; for example, `/dev/rdsk/dks1d457`.

Solaris

On Solaris, specify

- An absolute path name if you specify `UNIX` or `FILE` for *`type`*.
- The path name of the raw device if you specify `PARTITION` or `SLICE` as *`type`*; for example, `/dev/rdsk/c1t5d0s2`.

`{expandable | nonexpandable}`

Indicates whether the partition is expandable or nonexpandable in size. By default, raw partitions are nonexpandable and file partitions are expandable.

Modifying Partition Size

Expansion of the rawfs occurs differently, depending on whether the rawfs is in raw partitions or file partitions, as explained in the following:

- You can expand raw partitions only by setting aside additional raw space, adjusting the `Partition`*`N`* statements as needed, and then restarting the server.
- File partitions can expand dynamically if you specify the `expandable` attribute in the `Partition`*`N`* statement and there is room for expansion in the file system that contains the file.

For example, suppose the rawfs contains one raw partition (`/dev/rsd4x`) and three files (`/usr1/one`, `/usr2/two`, and `/usr3/three`). If you want to permit only two of the files to expand, your server parameter file might include

```
Partition0: UNIX /usr1/one expandable
Partition1: UNIX /usr2/two expandable
Partition2: UNIX /usr3/three nonexpandable
Partition3: PARTITION /dev/rsd4x
```

Increasing partition size

To increase the size of one or more existing partitions, check the sizes of the partitions with the appropriate operating system command. The new partition must be no smaller than the partition it is replacing. To copy the data from an existing small partition to a new larger one use `dd`(1). Then substitute each new name for the corresponding old one in the server parameter file.

Shrinking rawfs size

You cannot shrink the size of a rawfs by removing one of its partitions.

Adding partitions

Use the `osconfig` utility to establish your rawfs. You can use the following procedure to add partitions to the rawfs:

1. Make sure your raw partition is in place or that you created the file that will be your file partition.
2. Use the `ossvrshtd` utility to shut down the server. On the cluster operating system, use the `stop_osserver` script.
3. Edit the file `$OS_ROOTDIR/etc/`*`hostname`*`_server_parameters`. Add a `Partition`*`N`* for each new partition. See Specifying the Partitions in a Rawfs on page 303.
4. If you are using failover, you must do either of the following, depending on the type of partition you are adding:
   - *File-based partitions*: If you are adding a file-based partition and are using either ObjectStore-managed or cluster-managed failover, you must edit the Identical Pathnames on Failover Server parameter to include a string that matches the pathname of the partition. The `osserver` process will not start up unless you edit the parameter. For more information about this parameter, see Identical Pathnames on Failover Server.
   - *Raw-device partitions*: If you are adding a raw-device partition and are using cluster-managed failover, you must edit the cluster resource named `EXLNosserver-`*`hostname`*`-disks` to include the raw device in its `GlobalDevicePaths` property. Not adjusting the cluster resource will not prevent the `osserver` from starting, but it will slow down access to the new raw-device partition.
5. As `root`, restart the server with the `osserver` utility. On the cluster operating system, use the `start_osserver` script.

The server initializes only the new partitions. The content of existing partitions remains intact.

To reinitialize the entire rawfs, see Reconfiguring the Rawfs on page 37.

Setting Cache Manager Parameters

If a cache manager is not already running, it starts automatically when a client application starts.

You can specify the following cache manager parameters:

- `Cache Directory`
- `Commseg Directory`
- `Hard Allocation Limit`
- `Mount Table Pathname`
- `Preallocate Cache Files`
- `Soft Allocation Limit`

- `Temporary Files Permission`

If you modify a cache manager parameter, the parameter does not take effect until the cache manager is shut down and restarted.

Cache Directory Parameter

Default: `/tmp/ostore`

The `Cache Directory` parameter specifies the directory in which ObjectStore places the client cache file. This directory should not be an NFS mount point because this can result in slower client performance and can create the potential for problems with memory mapping over NFS. The default value is `/tmp/ostore`.

If the environment variable `OS_CACHE_DIR` is set in a client, ObjectStore uses that directory for that client. Cache and commseg files should be in the same directory.

Make sure that this directory exists. The cache manager does not create it automatically.

Commseg Directory Parameter

Default: `/tmp/ostore`

The `Commseg Directory` parameter specifies the directory in which ObjectStore places the communications segment. This directory should not be an NFS mount point because this can result in slower client performance and can create the potential for problems with memory mapping over NFS. The default value is `/tmp/ostore.`

If the environment variable `OS_COMMSEG_DIR` is set in a client, ObjectStore uses that directory for that client. Cache and commseg files should be in the same directory.

Hard Allocation Limit Parameter

Default: `0`

The `Hard Allocation Limit` parameter specifies the upper bound, in bytes, on the amount of disk space that the cache manager can allocate for its cache files and commseg files. If you try to exceed this limit, you receive an error message similar to the following at ObjectStore initialization time (for example, when you call `objectstore::initialize()` or open or create a database):

```
Cache Manager hard limit (NNNNN) exceeded by request for MMMMM bytes
of cache and/or commseg
```

This parameter allows you to prevent ObjectStore files from using too much disk space.

The default value of `0` means that no limit is enforced.

Mount Table Pathname Parameter

Default: `/etc/mnttab` , `/etc/mtab`

The `Mount Table Pathname` parameter specifies the path of the mount table. The mount table indicates the UNIX file systems that are mounted and where they are in your file system. For example, if `grizzly` has a directory named `/archive/two` and you have it mounted on your file system as `/arc2`, NFS uses the mount table to translate the file name `/arc2/x/y` to `/archive/two/x/y`.

HP-UX and System V

The `Mount Table Pathname` parameter instructs ObjectStore to use a mount table that is not in the default location. For HP–UX, Solaris, and System V platforms (except SGI), the default path name is `/etc/mnttab`; otherwise, it is `/etc/mtab`.

ObjectStore Technical Support expects that few users will ever need this parameter. It is provided for the very unlikely situation in which you have something in your default mount table that ObjectStore cannot parse.

Preallocate Cache Files Parameter

Default: `true`

The `Preallocate Cache Files` parameter controls the preallocation of cache and commseg files.

Note

This parameter is for use with full client-server ObjectStore applications only. To control preallocation on ObjectStore / Single applications, use the `OS_PREALLOCATE_CACHE_FILES` environment variable, as described in OS_PREALLOCATE_CACHE_FILES (UNIX Only) on page 121.

By default, ObjectStore preallocates cache files during client initialization. If the files are large (hundreds of megabytes or more), preallocation can cause a noticeable delay during initialization. To increase performance, set this parameter to `false` (`0`), preventing the up-front allocation of disk blocks to cache and commseg files. Instead, space is allocated for the files as needed. Setting this parameter to `true` (nonzero) restores the default behavior.

Caution

When running clients with the `Preallocate Cache Files` parameter set to `false`, you must ensure that the file system that contains the cache and commseg files never gets full. If the system is unable to allocate free space for the cache files, the client or server can crash.

Soft Allocation Limit Parameter

Default: `0`

The `Soft Allocation Limit` parameter specifies the suggested upper bound, in bytes, on the amount of disk space that the cache manager can allocate for its cache files and commseg files. An application is allowed to exceed this limit to run to completion; when it finishes, the cache manager automatically deletes the cache and commseg files if the soft limit has been exceeded. Doing so frees disk space but can make application startup slower by forcing ObjectStore to create new cache and commseg files.

The `Soft Allocation Limit` parameter allows you to prevent ObjectStore cache files from using too much disk space.

The default value of `0` means that no limit is enforced.

Temporary Files Permission Parameter

Default: `0660`

When the `Temporary Files Permission` parameter is set to an octal value (for example, 0666), the cache manager creates commseg and cache files with file permissions equal to that value. If you do not use this parameter, the cache manager creates new cache and commseg files with the permission `0660`.

Note that the cache manager file permissions setting can also be affected by the `umask` of the shell that started the cache manager. For this reason, you should set the `umask` to `0` when using the `Temporary Files Permission` parameter.

Cache Manager Parameter File Location

The cache manager parameter file is similar to the server parameter file in terms of location and internal format. ObjectStore searches the following locations in the following order and uses the first cache manager parameter file that it finds:

1. Pathname specified with the `-p` option to `oscmgr6`.
2. `$OS_ROOTDIR/etc/`*`host`*`_cache_manager_parameters`, where *`host`* is the name of the current host as returned by the `hostname` program. `OS_ROOTDIR` is an environment variable.
3. `$OS_ROOTDIR/etc/cache_manager_parameters`.

Cache Manager Parameter File Format

The cache manager parameter file has the same characteristics as the server parameter file:

- Each line in a parameter file contains one parameter and its value.
- Parameters have text names, such as `Cache Directory`.
- Each parameter name is followed by a colon (:), some white space (tabs or spaces), and a value (either numeric or text, depending on the parameter).
- Comment lines must contain a # as the first nonblank character.
- Case is not significant for parameter names. Embedded spaces, however, are significant.
- If the same parameter name appears twice in the parameter file, only the first occurrence is read.

Example of a Cache Manager Parameter File

A cache manager parameter file can include one or more parameter specifications. Following is an example of a cache manager parameter file:

```
Cache Directory: /support2/local/ostore/tmp
Commseg Directory: /support2/local/ostore/tmp
Mount Table Pathname: /new/mount/pathname
Hard Allocation Limit: 10000000
```

When to Increase the Size of the Cache

A larger cache size does not necessarily improve performance.

On UNIX, an ObjectStore cache is a memory-mapped file. Suppose that you have a system in which the size of this memory-mapped file is a significant percentage of the system's real memory. In this situation, an application that is performing memory mapping for a large file and subsequently doing a lot of processing that does not involve the memory-mapped file can suffer a much poorer performance than if the application had not memory mapped the file — that is, if the application used straight UNIX file I/O.

The reason for this is that overall system swapping is increased by the presence of the semiactive memory-mapped file. That is, the application is not using or reusing the cached database pages frequently enough to benefit from a large cache file.

An example of such an application is a program that populates a database by inserting a large number of objects into a collection. In this case, with the exception of the actual collection object, the pages in the cache are not being reused. Such a program benefits from having a small cache (for example, 2 MB) because the objects are not being revisited and overall UNIX system swapping is low.

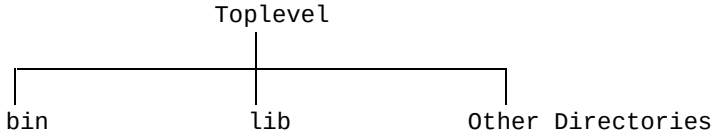
A program that would benefit from having a large cache is one that accesses a large number of objects (pages) and then reaccesses the same objects before replacing the objects (pages) in the cache with other objects (pages) fetched from the database. Of course, as the cache size starts to become a larger and larger percentage of the size of the machine's physical memory, more and more system swapping occurs and the performance gain of a large cache file begins to be lost.

The important issue with a large database is knowing how to properly cluster and segment it to achieve maximum performance. Properly designing a database so that it is clustered or segmented to achieve maximum performance is usually very application specific.

See `OS_CACHE_SIZE` on page 106 for information on setting the cache size.

ObjectStore Directory Structure

The ObjectStore directory structure appears in the following illustration. You must set `OS_ROOTDIR` to `Toplevel`.



Finding Files Containing ObjectStore Messages

When an ObjectStore daemon process sends output to `stdout` or `stderr`, ObjectStore routes the output to a corresponding file, as follows:

- Cache Manager: `/tmp/osc6_out`
- Server: `/tmp/oss_out`

If the file does not already exist, ObjectStore creates it; if the file does exist, ObjectStore appends the new information.

ObjectStore daemons seldom send messages to these files except in debug mode or under certain unusual error conditions. In debug mode, the ObjectStore server and cache manager send a lot of output to these files because all of the debugging output goes there. This information can be helpful in understanding and resolving an error. When you report a problem involving one of these daemons to Technical Support, find such a file, if it exists, and provide the contents.

When the daemon process is not running, you can safely delete the corresponding file. Usually, very little is ever sent to these files, so they are unlikely to occupy much disk space.

Identifying ObjectStore Database Files

On UNIX platforms you can edit the `/etc/magic` file so that the `file` command recognizes ObjectStore databases and ObjectStore backup and archive files. Add the following lines to the `/etc/magic` file:

```
0 string eXc\r\nelon\rdb\n! objectstore database
0 string ObJeCtStOrEdAtAbAsEbAcKuPiMaGe objectstore backup or archive image
```

Example

```
%file my_data.db
mydata.db: objectstore database
%
```

Using Tapes with the osbackup Utility

When you execute the `osbackup` utility, you can use standard UNIX tape drives, including quarter-inch cartridge and 8 mm cartridge drives. Some standard tape formats and capacities are shown in the following table:

Format	*Capacity in Megabytes*
QIC–11	30
QIC–24	60
QIC–150	150

Format	*Capacity in Megabytes*
QIC–525	525
EXB–8200	2200
EXB–8500	5000

The /tmp and /tmp/ostore Directories

ObjectStore uses the `/tmp` directory as the default location of the `oss_out` server output file and the `osc6_out` cache manager output file. For more information, see Finding Files Containing ObjectStore Messages on page 313.

ObjectStore uses the `/tmp/ostore` directory as the default location for the following files:

File	*For Information See*
Cache	`OS_CACHE_DIR (UNIX Only)` on page 106
Commseg	`OS_COMMSEG_DIR (UNIX Only)` on page 108
UNIX domain sockets	Modifying Network Port Settings on page 58

AIX Considerations

The following information provides additional considerations for ObjectStore on AIX.

Using SCSI Tape Drives

When using tapes with the `osbackup` and `osrestore` utilities, you have several choices for configuring SCSI tape drives as output devices on the RISC System/6000. These choices offer tradeoffs between convenience and performance.

You must choose values for these parameters:

- `Use Device Buffers` as set with `chdev` or SMIT
- Tape size given to the `osbackup` command
- Block size for the device as set with `chdev` or SMIT
- Block size given to the `osbackup` command

Device buffers

Writing a tape is much faster when you use device buffers. To do this, set `Use Device Buffers` to `yes` with `chdev` or SMIT. When you use device buffers, you must also specify a tape size to the `osbackup` command to ensure that buffered data is not

lost. Without a specified tape size, ObjectStore continues to write data until the end of the tape, not leaving room for any buffered data.

This situation does not occur when `Use Device Buffers` is set to `no`, and it is not necessary to specify a tape size to the `osbackup` command.

Block size

In general, you get the best tape usage by using variable-size blocks. To select variable-size blocks, you must use SMIT or `chdev` to set the *`block_size`* attribute to `0`.

The use of variable-size blocks on QIC–120 or QIC–150 media is apparently unique to IBM. If you write variable-size blocks on such a tape, you will not be able to read it on another system.

You can also use fixed-size records, which can facilitate tape duplication under certain circumstances.

To use fixed-size records, you must either set *`block_size`* in bytes in SMIT or with `chdev`, or use the `-b` control argument to `osbackup` to specify the block size in 512-byte sectors. Note that the `osbackup` command option overrides the SMIT setting, unlike utilities such as `tar`, where the SMIT setting takes precedence.

The block size must be 512 bytes or less or the `osbackup` utility cannot work.

Setting Up Permissions

When you have a client and a server on two different AIX machines, export the file systems as `setuid`. Also, add the switch

`-root=`*`server_host_name`*

to the `exportfs` command or to the line in the `/etc/exports` file. *`server_host_name`* is the name of the host of the ObjectStore server.

Failing to perform both steps can cause permission errors.

Troubleshooting Permission Denied Error

There is an AIX bug that occurs when the number of processes allowed per user is too low. You can list the current setting with the following command:

```
# lsattr -E -l sys0 -a maxuproc
maxuproc 40 Maximum # of processes allowed per user True
```

To increase the `maxuproc` parameter, enter the following command:

```
# chdev -l sys0 -a maxuproc=200
sys0 changed
```

When the number of processes allowed per user is too low, you might receive the following error:

```
No handler for exception:
permission to access this database was denied
re /usr/lpp/ostore/lib/liboscol.ldb while looking up file database
/usr/lpp/ostore/lib
/liboscol.ldb (host "aix_server1")
gmake: *** [AIX/os_schema.C] Error 1
aix_server1> ps -efl | grep ostore
```

```
260801 S     root 19379  3967   0  60 20 12892   516            Feb
23      -  0:00
/usr/lpp/ostore/lib/cmgr3 -AIX_SRC
200801 S pbergstr 32347 41739   1  60 20 11051    88  6052824 14:01:27
pts/15  0:00
grep ostore
262801 S pbergstr 40254  3967   0  60 20  9e3  1372    14:00:31     -
0:00 /
usr/lpp/ostore/lib/osserver -AIX_SRC
aix_server1>
```

Uninstalling ObjectStore

If you want to remove ObjectStore from your AIX system completely, use the following procedure:

1. Shut down the cache manager (`$OS_ROOTDIR/bin/oscmshtd`).
2. Shut down the server (`$OS_ROOTDIR/bin/ossvrshtd -f` *`hostname`*).
3. Use `ossvrchkpt` to ensure that all data is copied from the transaction log to the database that was changed.
4. Check the files `$OS_ROOTDIR/etc/`*`yourhostname`*`_server_parameters` and `$OS_ROOTDIR/etc/`*`yourhostname`*`_cache_manager_parameters` to see where your transaction log and temporary files are going and then remove them.
5. Remove the install area (`/usr/lpp/ODI` by default).
6. Remove links to `/usr/bin` if you created them with `osconfig`. Do not remove `/usr/bin/oslevel`.
7. Remove links to `/usr/lib` if you created them (`/usr/lib/libos*`).
8. Remove the link to `$OS_ROOTDIR/include` if you created it (`/usr/include/ostore`).
9. Remove `/etc/rc.objectstore`.

Chapter 8
Managing ObjectStore on Windows

This chapter provides information for managing ObjectStore on Windows platforms. For complete information, you should consult the first six chapters of this book along with this chapter.

The topics covered include the following:

Using ObjectStore Utilities

During installation, you use the ObjectStore setup utility (`setup.exe`) to configure ObjectStore daemons and applications — the server, cache manager, and the ObjectStore utilities.

All utility executables are stored in ObjectStore's `bin` directory.

On Windows, the `setup.exe` program installs the server and cache manager daemons as Windows Services. These services start automatically when the system boots. You can use the utility program `ossvrshtd` to stop the server. You can also start and stop ObjectStore services by using the Administrative Tools | Services console for 2000 and XP which is accessed from the Control Panel.

On Windows platforms, run ObjectStore utilities in a command prompt window.

Specifying File Database Path Names

You specify a file database with an operating system path name. For example:

```
os_database::open("d:\\carter\\myfiledb")
```

UNC path name

The syntax for specifying a UNC path name is

```
os_database::open("\\\\servername\\sharename\\myfiledb")
```

In C++, you must escape the backslash (\) with another backslash.

You can also specify a relative path name.

Server-remote databases

You usually store a file database on a host that is running an ObjectStore server. However, you can use a locator file to allow databases that are remote from the server. See Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 265.

Server names

You can specify a server name when you open or create a database. The server name identifies the server host on which the database is located. For example:

- Server on a UNIX host: `elvis:/usr/barbar/employees`
- Server on a Windows host: `elvis:D:\usr\barbar\employees`

This specifies a database, `employees`, in the directory `\usr\barbar` on the host `elvis`.

This method of specifying a path name is called *server-relative*. The token before the first colon names a server host and the rest of the string is parsed by that server in the syntax used on the server host operating system. Among other things, this is useful when no remote file protocol (such as NFS) is in use between the client and server hosts.

ObjectStore supports multibyte encoded path names. This includes environments where ObjectStore applications and servers are located on different machines and the client and server machines use different multibyte encodings.

Setting Server Parameters

On Windows, ObjectStore stores the parameters used by the server and cache manager in the Windows registry database.

ObjectStore server parameters are described in Chapter 2, Server Parameters, on page 77.

You can change these parameters by using the ObjectStore utility `setup.exe`. To set server parameters, run `setup.exe`, select `Advanced Options`, then select `Server Parameters`. The ObjectStore installation guide contains additional information about using `setup.exe`.

After you modify a server parameter, you must shut down and restart the server for the parameter to take effect.

Starting the Server

There are several ways to start the server. One way is to use the ObjectStore setup utility, `setup.exe`, to configure the server to start automatically at system startup.

On Windows, you can configure the server as a service. This is the preferred method.

When starting the server as a service, you can specify any of the `osserver` command-line options except `-con` or `-console`. You must specify them as Start Parameters in the Administrative Tools | Services console for 2000 and XP.

You can also run the server as a console application. In this case, you run `OSSERVER.EXE` from a command window, passing the `-console` option as the first argument.

At startup, ObjectStore configures the server according to parameter options in the registry database.

Ordinarily, you use server parameters to control the server's behavior. However, you can also specify the command-line options to `osserver`. For more information about the `osserver` utility and its options, see osserver on page 217.

Troubleshooting ObjectStore on Windows

You might have a situation in which you cannot run the server and the cache manager as services, even after defining autostart for them. You receive the following error:

```
Starting process
Process could not start
error 1067: Process terminated unexpectedly
```

The first thing to assess is whether the server and cache manager can be started in console mode by using the `-con` option of `osserver`. If they start properly in console mode, the problem is likely to be a permission problem or an incorrect definition for the ObjectStore image path in the registry.

Try the following to determine what is wrong with the ObjectStore definition of Windows Services:

1. If you ran an earlier ObjectStore release, confirm that it was properly uninstalled before installing the new release. This is necessary so that the image path is properly set for the Windows Services.
2. Ensure that ObjectStore was installed using an account with Administrative privileges.
3. To run the server and cache manager as Services, you must log on with Administrative privileges.
4. Confirm that the `OS_ROOTDIR` environment variable is set and that it points to the proper ObjectStore installation.

5 Make sure that the image path for the server and cache manager is correct. To check this, run `Regedt32` from a window and follow the path

```
HKEY_LOCAL_MACHINE | System | CurrentControlSet | Services
| ObjectStore Cache Manager R6.0
```

Then verify that the setting for the image path is correct. Do the same for `ObjectStore Server R6.0`.

6 Verify that the user account used to log in includes the `log in as a service...` privilege.

If the same executables do not bring up ObjectStore properly even in console mode (especially if the server does not start from the command line when you use the `-con` option), it is not a permission problem.

If it is not a permission problem, it is very likely that the transaction log has not been initialized. In this case, a message can be seen in the `%OS_TMPDIR%\osserver.txt` output:

```
When a partition is not specified, the transaction log is needed.
```

This is usually the last message in `osserver.txt` or in the server startup output in debug mode. If this is not the case, in the window from where you start the server, enter

```
>set OS_DEBUG_NETWORK=1
```

and

```
>start /min osserver -con -F -v -d 10 > server.out
```

or

```
>osserver -con -F -v -d 10 > server.out
```

if the `start` command is not recognized.

After that, send the `server.out` file to Technical Support.

Creating a Rawfs

Maintaining a rawfs provides a fast, convenient way to manage all ObjectStore databases at your site. See Managing the Rawfs on page 35. On Windows platforms, you can use the following rawfs configurations:

- File partitions
- Disk partitions
- Physical disks

If you want to use ObjectStore's high-availability features, you need to configure the rawfs to use either disk partitions or a physical disk. In either of these configurations, you should first create the disk partitions with a disk management utility, such as the Windows Disk Management snap-in, before specifying them for use by the rawfs.

The Disk Management snap-in is available from the Windows Start menu under Administrative Tools | Computer Management.

Using setup.exe to Add Rawfs Partitions

You use the ObjectStore `setup.exe` utility to create and modify rawfs. Be sure to shut down the server before you run `setup.exe`. After you specify server parameters, `setup.exe` provides the opportunity to set up a rawfs. If you are configuring a rawfs to use file partitions, you specify the name and size of the file partition and whether or not it is expandable.

If you are configuring a rawfs to use disk partitions or physical disks, you select the partitions from a list of available partitions.

Modifying Partition Size

Before you modify your rawfs configuration, back up your data by using the `osbackup` utility. See osbackup on page 143.

Rawfs expansion

Expansion of the rawfs occurs differently, depending on whether the rawfs is in raw partitions or file partitions:

- You can expand raw partitions only by setting aside additional raw space, adjusting the `Partition`*`N`* statements as needed, and then restarting the server.
- File partitions can expand dynamically if you specify the `expandable` attribute in the `Partition`*`N`* statement and there is room for expansion in the file system that contains the file.

For more information about modifying a rawfs, see Reconfiguring the Rawfs on page 37.

Starting the Cache Manager

The cache manager is a Windows Service that is usually configured to start when the system is booted. Alternatively, you can configure it to start automatically when it is needed. Use the Administrative Tools | Services console for 2000 and XP to configure the cache manager for manual startup.

To start the cache manager from your account, you must have `domain user` listed as one of your privileges. If you do not, you might receive the message

```
Error auto starting Cache Manager. Unable to open service database.
Access is denied.
```

You might receive this message when you log in to your domain account rather than logging directly into the computer name account. If this is the case, you need to ask your system administrator to check your user privileges in that domain or configure the cache manager for automatic startup.

If you want to start the cache manager as a console application, use the following command:

```
oscmgr6 -con [ -d debug_level ]
```

Finding Files Containing ObjectStore Messages

In some cases, when an ObjectStore daemon process reports an error, ObjectStore routes the output to a file. These files are

- Server errors: `OSSERVER.TXT`
- Cache manager errors: `OSCMGR6.TXT`

If the file does not already exist, ObjectStore creates it; if the file does exist, ObjectStore appends the error output to it.

ObjectStore uses the path assigned to the environment variable `OS_TMPDIR` to determine the directory in which to place these files. If `OS_TMPDIR` is not set, ObjectStore uses the path returned by the Win32 API `GetTempPath()`, which is controlled by the `TMP` and `TEMP` environment variables.

ObjectStore daemons seldom send messages to these files except in debug mode or under certain unusual error conditions. In debug mode, the ObjectStore server and cache manager send a lot of output to these files because all of the debugging output goes there. This information can be helpful in understanding and resolving an error. When you report a problem involving one of these daemons to Technical Support, find such a file, if it exists, and provide the contents.

When the daemon process is not running, you can safely delete the corresponding file. Usually, very little is ever sent to these files, so they are unlikely to occupy much disk space.

Accessing UNIX Databases from Windows

When accessing UNIX file databases over a network (such as with Intergraph PCNFS), the ObjectStore client on Windows prompts for a name and password. See `Authentication Required` on page 80.

If you want the ObjectStore client to use RPC authentication, use the Windows `REGEDT32` utility and set the following variables in the registry, where *`username`* is your username:

```
HKEY_LOCAL_MACHINE\Software\Object Design Inc.\
  ObjectStore 6.0\Remote\username\UNIX.UID
HKEY_LOCAL_MACHINE\Software\Object Design Inc.\
  ObjectStore 6.0\Remote\username\UNIX.GID
```

Set these variables (which are strings, such as REG-SZ) to the numeric values of the UNIX user and group IDs required. Note that you can edit these values only from a Windows account with Administrator privileges.

Note that the registry location

```
HKEY_LOCAL_MACHINE\Software\Object Design Inc.\
  ObjectStore 6.0\Remote
```

has Administrator rights only. This is to prevent a security breach that can result if ordinary users have write privileges to this area. If you set up the values for an ordinary user who does not have Administrator privileges and the user tries to run an ObjectStore program, the user must still enter a user name and password to access the server.

To prevent this, you can change the permissions of the registry location

```
HKEY_LOCAL_MACHINE\Software\Object Design Inc.\
  ObjectStore 6.0\Remote
```

to have read-only privileges for users with non-Administrator privileges. You can set the privileges by using the `Security | Permissions` dialog in the registry editor.

About Client/Server Communication

Windows clients and servers can use two network layers:

- Within a single machine, ObjectStore uses an interprocess communication mechanism based on named shared-memory objects.
- Windows Sockets provide TCP/IP connections.

A server can and typically does serve both networks simultaneously. A client chooses the first network available that recognizes the name of the server host.

For more information, see Network Support in *Installation for Windows*.

Using an ObjectStore Server on Windows to Access Remote Databases

Netware

ObjectStore can be used with the Gateway Services for NetWare (GSNW) feature.

Allow Remote Database Access

If a server is configured to allow remote databases, the server can use UNC paths to access such databases. Using UNC paths is the preferred method because of permissions issues surrounding the use by a service of mounted drives. Server-relative paths such as

```
serverhost:\\filehost\sharename\directory\foo.odb
```

or locator files that redirect all UNC paths to a named server host take advantage of this feature. In general, it is unnecessary to

- Mount file host drives on your server host
- Use `REPLACE` statements in locator files

Use this feature with 2000/XP as well as with NetWare file hosts.

Access Control for Remote Databases

Windows NT does not allow the server to use credentials obtained from a remote client host on a remote file host. There are two alternatives for dealing with this issue. They are described in the Microsoft Knowledge Base article Q132679, available on the Microsoft TechNet CD or at the Microsoft Web site

```
http://www.microsoft.com
```

The first choice is to run the server normally, as *Local System*. This requires that the user *Everyone* is able to access remote files on behalf of remote clients. Note that this solution requires you to create an account for *Everyone* on non-Windows NT systems.

The second choice is to run the server as another user. You can set this up by means of the Control Panel Services applet.

Changing the Registry Location for ObjectStore (Windows Only)

ObjectStore uses the Windows registry for different operations, such as looking up parameters or the UNIX user and group IDs. When you install ObjectStore, the Windows registry location is set to the following:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore 6.0
```

You can, however, change the registry location that you want ObjectStore to use. Changing this location is especially useful when you have embedded ObjectStore in an application. By specifying a different location, you can prevent another application that also uses ObjectStore from overwriting information in the registry location used by your application.

Server

To specify a different registry location, use the `-r` option to the `osserver` utility, as explained in osserver on page 217. This option takes a string argument that specifies the registry location that you want `osserver` to use, relative to `HKEY_LOCAL_MACHINE\SOFTWARE`.

To pass the `-r` option and its location argument to `osserver`, call the NT system call `CreateService()` when installing your application. One of the arguments to `CreateService()` is *`lpBinaryPathName`*. You can set this argument to the path of `osserver`, along with the -r option and its argument. The path of `osserver` must be enclosed by double quotes, and the quoted pathname must be followed by the `-r` option and its argument. This embedded argument is a string that specifies the registry location that ObjectStore is to use. Here is an example of the *`lpBinaryPathName`* argument:

```
const char* lpBinaryPathName =
  "\"d:\\odi\\ostore\\bin\\osserver.exe\" -r my_application";
```

After calling `CreateService()` with this argument, if you were to navigate to the `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\`*`service_name`* registry key and look at the value associated with the `ImagePath` key, you would find

```
"d:\\odi\\ostore\\bin\\osserver.exe\" -r my_application
```

At this point, the ObjectStore server can look in the following registry location for its parameters:

```
HKEY_LOCAL_MACHINE\SYSTEM\my_application\Server
```

Cache Manager

If you also want the ObjectStore cache manager to use a different registry location, you can follow a similar procedure to change its location. The cache manager (`oscmgr6`) also has a `-r` option that allows you to specify a different location. You can use the `CreateService()` call to pass the option and its argument to the cache manager. After `CreateService()` has been called with the argument

```
const char* lpBinaryPathName =
  "d:\\odi\\ostore\\bin\\oscmgr6.exe\" -r my_application
```

the cache manager can look in the following registry location for its parameters:

```
HKEY_LOCAL_MACHINE\SYSTEM\my_application\Cache Manager
```

Clients

On the client side, you can set the registry location programatically by calling `os_authentication::set_nt_registry_location()` or by setting the `OS_REMOTE_AUTH_REGISTRY_LOCATION` environment variable. , If the location were set to `my_application`, clients would look in the following registry location for authentication information:

```
HKEY_LOCAL_MACHINE\SYSTEM\my_application\Remote
```

Note that, if `set_nt_registry_location()` and `OS_REMOTE_AUTH_REGISTRY_LOCATION` specify conflicting locations, the environment variable takes precedence. For more information, see the following:

- `os_authentication::set_nt_registry_location()` in *C++ API Reference*
- `OS_REMOTE_AUTH_REGISTRY_LOCATION (Windows Only)` on page 122

Index

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

W