OBJECTSTORE

MANAGEMENT

RELEASE 5.1

March 1998

ObjectStore Management

ObjectStore Release 5.1 for all platforms, March 1998

Object Design, Inc.
Twenty Five Mall Road
Burlington, MA 01803-4194

Part number: SW-OS-DOC-MGT-510

Contents

Preface

Purpose

ObjectStore Management provides information needed to perform management tasks on ObjectStore Servers and clients. This book supports ObjectStore Release 5.1.

Audience

There are two audiences for this book:

- Administrators responsible for keeping ObjectStore running, and doing such tasks 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 half of this book provides information that applies to all ObjectStore platforms. The second half contains platform-specific chapters. For complete information, you must read the general chapters along with the chapter for your platform.

Notation Conventions

This document uses the following conventions:

Convention	***Meaning***
Bold	Bold typeface indicates user input or code.
Sans serif	Sans serif typeface indicates system output.

Convention	*Meaning*
Italic sans serif	Italic sans serif typeface indicates a variable for which you must supply a value. This most often appears in a syntax line or table.
Italic serif	In text, italic serif typeface indicates the first use of an important term.
[ ]	Brackets enclose optional arguments.
{ *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 indicate that you can repeat the immediately previous item. In examples, they also indicate omissions.
	Indicates that the operating system named inside the circle supports or does not support the feature being discussed.

ObjectStore Release 5.1 Documentation

The ObjectStore Release 5.1 documentation is chiefly distributed online in Web-browsable format. If you want to order printed books, contact your Object Design sales representative.

Your use of ObjectStore documentation depends on your role and level of experience with ObjectStore. You can find an overview description of each book in the ObjectStore documentation set at URL **http://www.objectdesign.com.** Select **Products** and then select **Product Documentation** to view these descriptions.

Internet Sources of More Information

World Wide Web

Object Design's support organization provides a number of information resources. These are available to you through a Web browser such as Internet Explorer or Netscape. You can obtain information by accessing the Object Design home page with the URL **http://www.objectdesign.com.** Select **Technical Support**. Select **Support Communications** for detailed instructions about different methods of obtaining information from support.

Internet gateway

You can obtain such information as frequently asked questions (FAQs) from Object Design's Internet gateway machine as well as from the Web. This machine is called **ftp.objectdesign.com** and its Internet address is 198.3.16.26. You can use **ftp** to retrieve the FAQs from there. Use the login name **odiftp** and the password obtained from **patch-info**. This password also changes monthly, but you can automatically receive the updated password by subscribing to **patch-info**. See the **README** file for guidelines for using this connection. The FAQs are in the subdirectory **./FAQ**. This directory contains a group of subdirectories organized by topic. The file **./FAQ/FAQ.tar.Z** is a compressed **tar** version of this hierarchy that you can download.

Automatic email notification

In addition to the previous methods of obtaining Object Design's latest patch updates (available on the **ftp** server as well as the Object Design Support home page), you can now automatically be notified of updates. To subscribe, send email to **patch-info-request@objectdesign.com** with the keyword **SUBSCRIBE patch-info <***your siteid***>** in the body of your email. This will subscribe you to Object Design's patch information server daemon that automatically provides site access information and notification of other changes to the online support services. Your site ID is listed on any shipment from Object Design, or you can contact your Object Design sales administrator for the site ID information.

Training

If you are in North America, for information about Object Design's educational offerings, or to order additional documents, call 781.674. 5000, Monday through Friday from 8:30 AM to 5:30 PM Eastern Time.

If you are outside North America, call your Object Design sales representative.

Your Comments

Object Design welcomes your comments about ObjectStore documentation. Send your feedback to **support@objectdesign.com**. To expedite your message, begin the subject with **Doc:**. For example:

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

You can also fax your comments to 781.674.5440.

Chapter 1
Overview of Managing ObjectStore

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

- With the ObjectStore C++ interface, create and modify C++ objects (as well as C structs) instead of tables, columns, rows, and tuples
- Access data in the same format in which it exists in the application
- 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

What Is an ObjectStore Database?

An ObjectStore database is a storage location for persistent objects.

Segments

A database contains *segments*, which are variable-sized regions of memory. Each segment is made up of pages. The unit of transfer from persistent storage (an ObjectStore database) to program memory can be a page, a number of pages, or a segment.

Default segment

When an application creates a database, the database automatically has a special segment, called the *default segment*, for storing your data. The application can be written to create additional segments if that is required. If the application does not specify a segment when storing data, ObjectStore always stores data in the default segment. Segments grow (add pages) to accommodate the data that is stored in them. The size of a segment is platform dependent. Size is limited by the available persistent address range, which is typically between 230 MB and 2 GB.

Segment location

It is not possible to specify segments at fixed locations. You specify the pathname for a database and ObjectStore determines where to locate the segments.

Number of pages in a segment

You do not need to specify how many pages are in a segment. The default segment, and any segments that an application creates, add pages as needed to hold their data. Pages are a fixed size that depends on your operating system; 4K is a common size. When an application specifies a new segment, ObjectStore creates it on a page boundary.

Page size

Page size does not limit the size of an object that you can store. Storage of an object is independent of page size. Many objects can exist on one page. One object can span many pages.

What Kinds of Databases Are There?

You can use ObjectStore to store objects in two kinds of databases, *file databases* 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 ObjectStore utilities described in this book. A file database has a standard operating system pathname.

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 like the native file system. It can include links, but each link must be to another ObjectStore rawfs. 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 methods for operating on databases and directories in a rawfs. An ObjectStore Server can manage one rawfs, which consists of one or more Server partitions.

Rawfs partitions

You specify each partition in the rawfs with a **Partition***n* statement in the Server parameter file. Each partition can be either

- 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.

  On AIX, Digital UNIX, SGI IRIX, and Solaris, a raw partition can be greater than two gigabytes. On HP–UX, a raw partition can be as large as four gigabytes. On Windows, a raw partition can be larger than four gigabytes.

- 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 pathname. 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.

Rawfs database name format

ObjectStore recognizes a rawfs database by the following format:

hostname`::/`*database_pathname*

A double colon separates the host name from the database pathname. 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 pathnames identify different databases:

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

You can create a rawfs during ObjectStore installation or at any time after installation. See the chapter specific to your platform for instructions.

How Do Objects Get into a Database?

When an application allocates an object in persistent storage, it specifies the database to contain that storage. An application can create a database with a call to one of the member functions listed below. You name the database in a pathname argument to the function that creates the database.

- **os_database::create()**
- **os_database::lookup()**
- **os_database::open()**

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

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 of the processes 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.

Rawfs management

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.

Usually, 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 281, 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.

Multiple Servers on a host

A host can run one Server of a given ObjectStore release. You can run two ObjectStore Servers on the same host if they are different versions of ObjectStore. For example, you can run a Release 5 and Release 4 Server on the same host. Start them on different ports and use the ports file to let clients know which one to contact. See Modifying Network Port Settings on page 51 for details.

A network can have a number of Servers.

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

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 physical memory. When a client application needs an object stored in a database, if the page that holds the object is

- Already in physical memory or the cache, the application just accesses it.
- Not in physical memory and not in the cache, or
  - In the cache but not yet accessed
  - In the cache but previously accessed with different permissions (for example, read-only instead of update)

  then the application receives a page fault; the ObjectStore client requests the page from the Server, puts it in the cache, and continues with program execution.

To change the default size of the client cache, use the **OS_CACHE_SIZE** environment variable. See **OS_CACHE_SIZE** on page 98.

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

UNIX

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 and OS/2

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.

See also

Refer to the *ObjectStore Technical Overview* for additional information.

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 do not need to configure ports.

The name of the Cache Manager executable is **oscmgr4**.

On UNIX, the **oscminit** executable starts **oscmgr4**. Normally, you never need to invoke **oscminit**.

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 if the client requesting permission must wait.

If you are running an ObjectStore application that uses a database that nobody else is using, there are no callback messages for that database.

For information about ownership and locks, see Callback Messages Background on page 58.

Commseg

The *commseg* (communications segment) is where the Cache Manager maintains information about permissions on pages and whether or not 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.

You can use the following environment variables to modify the default attributes of the commseg. See **OS_COMMSEG_RESERVED_SIZE** on page 101 for details.

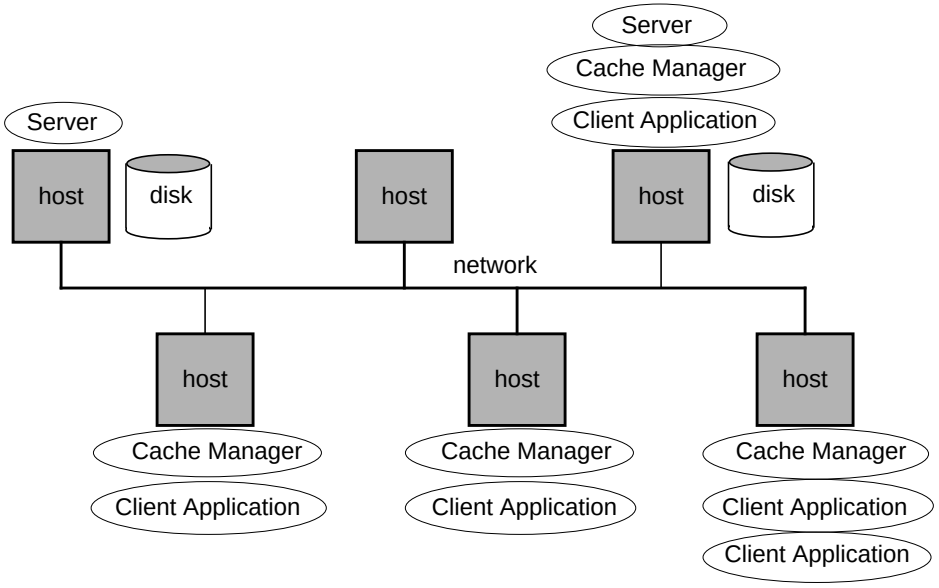
OS_COMMSEG_RESERVED_SIZE	Maximum size of the commseg
OS_COMMSEG_SIZE	Size of the commseg
OS_COMMSEG_START	Starting address of the commseg (rarely specified; ObjectStore usually determines this)

UNIX

By default, ObjectStore places the commseg file in the **/tmp/ostore** directory. You can specify an alternate 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 and OS/2

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 illustration that follows shows the ObjectStore processes specific to Windows and OS/2.


Server
Cache Manager
Client Application
Server
host
disk
host
host
disk
network
host
host
host
Cache Manager
Client Application
Cache Manager
Client Application
Cache Manager
Client Application
Client Application

Managing Processes

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

- Server (**osserver**)
- Client (application)
- Cache Manager (**oscmgr4**)

In general, the actions you perform on ObjectStore processes are

- Start-up
- Shutdown
- Obtain process status information

Communication Among ObjectStore Processes

The following table summarizes the communication among the Server, client, and Cache Manager processes.

Server	Responds to client requests for pages
	Sends callback messages to Cache Manager to request locks 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 Server
Cache Manager	Receives callback requests from 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. Normally, you do not need to modify network connections. However, if you do need to make changes, see Modifying Network Port Settings on page 51.

Starting ObjectStore Processes

Server

The chapter supporting your platform provides instructions for starting the Server on your platform. Usually, 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 51.

There are many Server parameters you can set to determine the behavior of the Server. Chapter 2, Server Parameters, on page 69, 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 NT

On Windows NT, the Server and Cache Manager normally run as NT Services. You can use the Services applet in the Control Panel to start and stop the Server and Cache Manager and to determine whether or not they start automatically when you boot the system.

Stopping ObjectStore Processes

Server

You need to shut down the Server

- Before you reboot or halt the host
- After you modify Server parameters
- To resize the transaction log (see Log File Size on page 18)
- To add a partition to the rawfs

Use the following steps to shut down the Server:

1 Use the **ossvrstat** utility to determine if 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** method. See ossvrstat: Displaying Server and Client Information on page 261.

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 that are 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: Disconnecting a Client Thread on a Server on page 254.

4 Use the **ossvrshtd** utility to shut down the Server. See ossvrshtd: Shutting Down the Server on page 259.

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: Disconnecting a Client Thread on a Server on page 254 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: Displaying Cache Manager Status on page 160. Use the **oscmshtd** utility to shut down the Cache Manager. See oscmshtd: Shutting Down the Cache Manager on page 159 for details. The next client process that starts automatically starts the Cache Manager.

Obtaining Process Status Information

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. See Finding Files Containing ObjectStore Messages in the chapter corresponding to your platform.

ObjectStore daemons seldom send messages to these files except under certain unusual error conditions. In these cases, this information can be helpful in understanding and resolving an error. When you report to Object Design a problem that might involve one of these daemons, find such a file if it exists and provide the contents.

Meters

The **ossvrstat** utility displays information from a Server. For example:

- How much data the Server sent to clients
- How much modified data clients sent to the Server
- How many committed transactions the Server knows about
- How many times the Server chose a deadlock victim
- How many times a message from a client to the Server had to wait to use a message buffer
- How much data is 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: Displaying Server and Client Information on page 261.

Use the **oscmstat** utility to obtain information about the Cache Manager, client cache files, and commseg files. See oscmstat: Displaying Cache Manager Status on page 160.

Debugging Cache Manager

Use the following command line to start the Cache Manager in debug mode:

oscmgr4 0 *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 any problems with a Server, the first thing to do is run **ossvrping** to see if the Server is running. See ossvrping: Determining If a Server Is Running on page 258.

Monitoring the Server

You can use the **ossvrdebug** command to set the debug trace level for the Server. See ossvrdebug: Setting a Server Debug Trace Level on page 256 for more information.

To enable the **ossvrdebug** command, type **ossvrdebug -d 5**. To disable the command type **ossvrdebug -d 0.**

You can also 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.

To run the Server in debug mode, you must shut down and restart the Server with the debug option. See the chapter supporting your platform for specific details. When you no longer need the debug output, shut down the Server and restart it without the debug option.

Debug mode slows down the Server but does not affect clients. Use debug mode only when you are experiencing a problem.

Description of the Server Transaction Log

Each Server keeps a *transaction log*, which is 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 records about modified pages of data.

Log file

The log file stores database modifications until they are propagated to the database. The log does not have a consistent 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: Displaying Server and Client Information on page 261.

Location

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 pathname for the transaction log. See **Log File** on page 83. You cannot place the log file in a raw partition. Server performance is always better when the log is in the rawfs rather than in a native file.

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 accidentally deleted by users. Deleting the log file can cause database corruption.

Transactions, commits, and the log

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.

Log File Terms

The log file consists of two *log record segments* and a *log data segment*. The Server uses both the log record segments and the log data segment to store data.

Log record segment

The Server writes log records, such as commit records, to a log record segment. When a log record segment fills up, the Server switches to the other log record segment.

Log record buffer

The Server uses the log record buffer to form log records across sectors. Its size defines the maximum size that you can write to a log segment in one write operation. Each sector includes log header information that indicates which log blocks are valid.

When choosing the log record buffer size, you should consider both the cost of formatting data into a log record and the cost of a separate write operation.

Log data segment

As part of a commit record in the log segment, the Server writes data returned as part of a commit. But this occurs only if the total data being committed is small enough to fit in the log record buffer. If it is not, the Server writes the data to the data segment.

Propagation

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.

Sector

A *sector* is a 512-byte disk block (two sectors equal one kilobyte). When you run the **ossvrstat** utility, ObjectStore displays many statistics in number of sectors.

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 82 for details.

When you run the **ossvrstat** utility, **Current log size**, in the list of Server parameters, displays the current size of the log. Usually, the log grows to a size that accommodates your application and then stops growing. There is usually no reason to monitor log size or 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 the steps below.

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 clears the log before it actually shuts down.

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. See the chapter discussing your operating system for details.

Another way to shrink the size of the log file is to reinitialize the Server. See the chapter supporting your platform for specific information.

Warning: Never use **-ReallocateLog** if the Server is in an inconsistent state. Call the ObjectStore Support group for assistance.

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 22 is devoted to effective memory management.

Disk space and network

Network communications and disk input/output are often 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.

It is impossible to predict how much disk space you need for a client application because it is dependent on the application. However, here are some ways that an ObjectStore application uses disk space:

- On UNIX systems, the default location for the cache and commseg files is **/tmp/ostore**. On Windows and OS/2, the cache is in virtual memory and the commseg is in shared memory. Both are in locations that the operating system determines. 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 22.

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.
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 / receives short messages.	Not intensive.

Managing 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
- Backing store
  - Backing store for persistent data is the client cache.
  - Backing store for transient data is swap space.

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 is typically larger than virtual memory.
- Virtual memory includes only addresses that currently contain application data, either in physical memory or in backing store on disk.

What Are the Pieces of Address Space? How Are They 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, it follows that each client has its own persistent address space, called the *persistent storage region*. There are two environment variables that control the size and address range of the persistent storage region. **OS_AS_SIZE** and **OS_AS_START** have different default values on different platforms. For example, the persistent storage region on a SPARCstation 10 running Solaris 2 is typically 400 MB. See **OS_AS_SIZE** on page 93 and **OS_AS_START** on page 94 for specific details.

Each client also has its own transient address space.

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 16 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 some other 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.

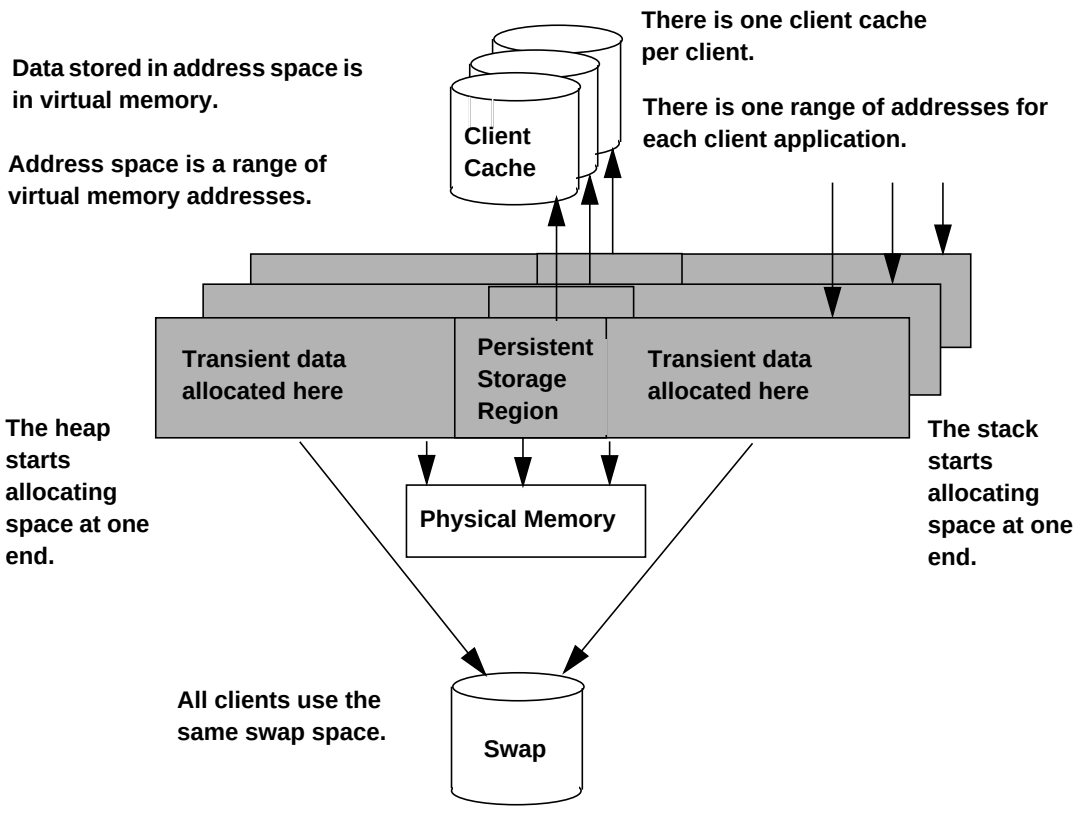
If the page contains transient data, the operating system pages it to swap space. Swap space is a disk file or disk partition that is shared by all applications running on the host.

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.



What Is the Relationship Between a Transaction and Address Space?

Address space limits the amount of data you can touch within the same top-level transaction. Address space must be reserved in order to correctly relocate data into virtual memory. As of Release 5, there are two address space assignment modes: immediate and deferred.

Under immediate assignment, the total amount of address space needed by a segment is reserved the first time any page within that segment is used in the transaction. This address space must be large enough to contain the segment itself, as well as the portions of other segments that are referred to by pointers within the segment.

Under deferred assignment, space is reserved as each page of the segment is used in the transaction, and the amount of space reserved by each page is the minimum required to correctly relocate the page into virtual memory.

The default behavior of ObjectStore in Release 5 is to use deferred assignment, since doing so prevents address space from being wasted. However, under certain circumstances, ObjectStore can detect that the pages in a segment can be brought into virtual memory without any pointer relocation. In order to ensure that the proper amount of address space is reserved without doing pointer relocation, ObjectStore uses immediate assignment for such a segment, provided that doing so does not increase the address space consumption in the transaction above half of **OS_AS_SIZE**.

There are several environment variables that can be used to change this behavior. See the sections on **OS_IMMEDIATE_THRESH**, **OS_MAX_IMMEDIATE_RANGES**, and **OS_FORCE_DEFERRED_ASSIGNMENT** for more information.

Also note that databases created prior to Release 5 or with **OS_FORCE_STANDARD_PRM_FORMAT** on can only use immediate assignment until they are upgraded using the **osupgprm** utility.

Normally, ObjectStore removes the assignments of persistent space at top-level transaction commit or abort. This allows the next top-level transaction to start with an empty persistent storage region.

However, if the application is using **retain_persistent_addresses**, ObjectStore does not release persistent address space until the program calls **release_persistent_addresses**.

Transaction commit

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, unless the number of pages exceeds the cache size
- Keeps in the cache pages already in the cache but not modified

Transaction abort

If a transaction aborts, the client

- Throws away modified pages because they are no longer valid

- Keeps in the cache pages already in the cache but not modified, in case they are needed in the next transaction

Guidelines

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

- Keeping transactions short.
- Using ObjectStore references rather than pointers. ObjectStore reserves address space for objects that are pointed to even if the transaction does not touch these objects. See *ObjectStore Advanced C++ API User Guide*, Using ObjectStore References, for further information.

What Happens When Resources Are Exhausted?

Address space is large and it is unusual to exhaust it. However, lack of enough of a particular address space piece affects the application.

Persistent address space

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

Physical memory

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

Client cache

When space in the client cache is needed but not available, ObjectStore attempts to migrate pages from the client cache back to the Server that supplied the data. This can impair performance. In some cases, this is impossible because the pages are wired into the cache. (ObjectStore determines which pages to wire (and unwire) into the cache. A page that is wired cannot be removed from the cache until it is unwired.) In this case, ObjectStore raises an internal error and aborts the transaction.

Swap space

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

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

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 assign 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 93.

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.

Client cache

Increasing the size of the cache can improve performance because it decreases the need to send pages containing persistent data back to the Server. You can control the size of the client cache with the **OS_CACHE_SIZE** environment variable. See **OS_CACHE_SIZE** on page 98.

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 how much swap space is available, for example, **pstat -s** in UNIX.

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?

How much 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.

Difference Between Assigning and Mapping an Address

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

When ObjectStore assigns an address to a page, it has determined where to put the page if the client needs it. The data on the page is not available to the client. Assigning an address reserves the address so that it cannot be assigned to another page. ObjectStore assigns address space in units of 64 KB, regardless of the page size on the platform.

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.

Summary of the Kinds of Memory an ObjectStore Application Uses

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 where you can store an application's persistent data. Each application has its own persistent memory.

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

Managing the Rawfs

The chapter for your operating system includes information for creating a rawfs. The advantages of a rawfs are that it

Advantages

- 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 the size of a disk. (*Large* usually refers to multigigabyte databases, but it also depends on how much disk space you have.)
- Allows applications to use ObjectStore in such a way that ObjectStore is invisible to end users.
- Simplifies management because all databases are in one location.

Security

- Provides greater security because the
  - Transaction log is hidden so there is no chance for accidental deletion.
  - 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.

Disadvantages

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 not as good as 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.

Performance

Performance varies depending on the type of database. The following lists the database types in order of performance, starting with the best:

1 Rawfs database stored in a raw partition

2 File database

3 Rawfs database stored in a file partition

OS_DIRMAN_HOST The **OS_DIRMAN_HOST** environment variable specifies a rawfs host name that ObjectStore places at the beginning of every pathname that does not already begin with the *host*:: rawfs prefix. This variable provides a convenient way to toggle between rawfs and native file systems. See **OS_DIRMAN_HOST** on page 105.

Utilities for Managing the Rawfs

ObjectStore provides the following utilities for managing the rawfs. There are many other utilities that you can use on the rawfs and on rawfs databases, but these are exclusively for operating on the rawfs.

- **oschhost** changes the host that a link in the rawfs points to.
- **osdf** shows the amount of used and available disk space for the rawfs on the specified Server.
- **osln** creates a symbolic link in the rawfs hierarchy.
- **osmkdir** creates a directory in the rawfs.
- **osrmdir** removes a directory from the rawfs.

Notes for Working with Rawfs Databases

Wildcards 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 on Server **oscar**:

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

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 **–** matches any character lexically between the pair, inclusive. If the first character following the opening [is a **!** 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 note On UNIX systems, you must precede the wildcard with a back slash (\) or enclose the pathname with the wildcard in quotation

marks (" "). This prevents the shell from interpreting the wildcard as a shell wildcard.

Reconfiguring the Rawfs

To reconfigure a rawfs, you must wipe out the existing rawfs.

See the chapter in this book discussing your platform. In that chapter are specific instructions for creating a rawfs and starting the Server on your platform. If you decide to reconfigure the rawfs, use that information with these general instructions.

Motivation

You might want to reconfigure a rawfs because you want to

- Shrink the size of the rawfs to save space
- Remove a partition

Procedure

Follow these instructions:

1. Confirm that you are on the machine you want to reinitialize and that your environment variables (**OS_ROOTDIR** and **OS_DIRMAN_HOST**) reflect 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. Use the **ossvrchkpt** utility to move everything in the log to the relevant databases.
4. Use the **osbackup** utility to back up the databases in your rawfs.
5. Use the **ossvrshtd** utility to shut down the Server.
6. Modify the **Partition**/**N** Server parameter specifications to reflect your new rawfs.
7. Initialize the Server by specifying **osserver -i**. This wipes out the rawfs partitions and starts the Server.
8. Use the **osrestore** utility (or **oscp**, if you used it when saving the database) to restore any needed databases.

Planning Your Configuration

This section provides information relevant to planning your configuration.

Mixing Network Protocols

You can mix network protocols among your Servers and clients. You can configure a Server to offer its services on as many types of networks as ObjectStore supports on that platform. Clients use the first available network that recognizes the name of the Server host.

For example, an OS/2 client running APPC and a UNIX client running TCP/IP can access the same Server.

However, be sure to avoid configurations that do not allow Servers to communicate with each other, as described in the next section.

All Servers in a Transaction Must Be Able to Communicate

You must ensure that when two or more ObjectStore Servers receive modified data in a single transaction, each of those Servers 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 usually 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.

Running an Application from a CDROM

You can place an ObjectStore application on a CDROM and run the application from the CDROM. You can access read-only databases on the CDROM. 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 CDROM.

You should validate the schemas before you place an application and/or database on a CDROM. In other words, run the application on the database in question (in update mode) and then put the application and/or database on the CDROM. 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 CDROM.

See additional schema validation discussions in the *ObjectStore C++ API User Guide*, in Chapter 2, Persistence.

What You Need to Know About the API

Some administrative tasks require familiarity with 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 *ObjectStore C++ API User Guide*.

For Information About	***See***
References	Chapter 2, Persistence
Collections	Chapter 5, Collections
Relationships	Chapter 6, Data Integrity

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, Transactions, in the *ObjectStore 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). See the *ObjectStore Advanced C++ API User Guide* for information about indexes — Chapter 5, Queries and Indexes, and for information about controlling locking granularity and lock caching — Chapter 3, Transactions.

Managing Databases

To manage databases, you are likely to perform the following typical operations:

Operation	*Utility and Reference*
Copy	oscp: Copying Databases on page 171
Move	osmv: Moving Directories and Databases on page 200
Delete	osrm: Removing Databases and Rawfs Links on page 222
Back up	osbackup: Backing Up Databases on page 139
Restore	osrestore: Restoring Databases from Backups on page 216
Dump	osdump: Dumping Databases on page 177
Load	osload: Loading Databases on page 196
Compact	oscompact: Compacting Databases on page 164

Additional utilities for operating on databases are documented in Chapter 4, Utilities, on page 127.

Pointers and references

Database management is complicated by the fact that a database often

- Points to other databases
- Is pointed to by other databases

Data set

To perform many administrative activities, you must know the relationships among the databases so you can identify the *data set*. The data set includes

- DatabaseA
- All databases that point to or reference databaseA
- All databases that databaseA points to or references

If a database contains no external pointers or references and is not pointed to or referenced by any other database, then it alone is the data set. The application developer is often the best source for this information. If the developer is not available, you can iteratively use the **ossize** utility to establish the data set. See ossize: Displaying Database Size on page 248.

Schema databases

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

Operate on the set

After you identify a data set, you can perform an administrative function on the entire set. If you rename or move an individual database in the set, use the **oschangedbref** utility described in oschangedbref: Changing External Database References on page 146 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: Verifying Pointers and References in a Database on page 274 to confirm that there are no bad pointers or dangling references. This utility detects inconsistencies; it does not fix them.

   If you are moving a database within a rawfs, verification is not necessary because the move just changes the name.

2 Run the **ossize** utility described in ossize: Displaying Database Size on page 248 to ensure that cross-database references are correctly established.

3 Perform steps 1 and 2 on all databases in the data set.

4 Run the **ossvrchkpt** utility described in ossvrchkpt: Moving Data Out of the Server Transaction Log on page 253 to ensure that all data has been propagated from the log to the affected database.

Avoiding difficulties

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.

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 281, for details.

Overview of the Backup/Restore Facility

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

- **osbackup** and **osarchiv**

  These utilities back up databases you specify to a secondary storage location. The **osbackup** utility performs backups according to the level you specify; that is, it copies segments 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 also 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: Backing Up Databases on page 139) according to a schedule that you determine.

2 Keep the **osarchiv** utility active. See osarchiv: Logging Transactions Between Backups on page 132.

3 Run the **osrestore** utility (described in osrestore: Restoring Databases from Backups on page 216) to recover data backed up by the **osbackup** utility.

4 Run the **osrecovr** utility (described in osrecovr: Restoring Databases from Archive Logs on page 206) to recover data from archive logs.

On-Line Backup and Archive Logging

On-line 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 needed to recover modifications made after a backup. On-line backup gives you a starting point for using archive logging.

On-line 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, allowing transaction-consistent backups of interdependent distributed databases. You can back up databases on different Servers in the same backup operation.
- Concurrent read / write access by ObjectStore applications to databases in the backup set. That is, on-line 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, allowing 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 *ObjectStore Building C++ Interface Applications*, Chapter 5, Building Applications for Use on Multiple Platforms.)
- 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.

On-Line 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 on-line, ObjectStore applications cannot access databases that are being restored until the entire restoration process has completed.

On-line restore and recovery offer these features:

- You can always recover a committed transaction if archive logging was being performed for the database when the transaction was committed.
- 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 41 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:

- 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 paragraphs that follow provide suggestions and examples of methods and tools you might use, depending on your answers to the questions posed.

General Backup Practices

Object Design, Inc. recommends a nightly backup of the ObjectStore database at a minimum. The **osbackup** command is an on-line utility that runs as if it were an ObjectStore MVCC client. It does not cause any concurrency conflicts.

WIN

You might choose to run Microsoft's Schedule service or use the **at** command to automatically invoke this operation each night. 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 and protects you from a disk crash.

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 production size databases be estimated. If the databases are not in the gigabyte range, then 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:

Sunday:	Backup level 0
Wednesday:	Backup level 1
Mon. Tues. Thurs. Fri. Sat.:	Backup level 2

For example, if you need to restore the databases on Friday morning, then you must restore the backup images from Sunday, Wednesday, and Thursday. It is advisable to save the previous level 0 backup in another spot before starting the new level 0 backup.

osbackup command flags

Here is a sample backup command you might use:

osbackup -i bkup_rec.txt -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 which databases have been backed up, and when they were backed up. The **osbackup** utility uses this information to determine which segments within a database have been modified since the last backup at a lower level. The utility then backs up only modified segments. 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 segment level, meaning that a segment 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 segments in all specified databases.
- The **-f** flag specifies the location of the backup image.

See osbackup: Backing Up Databases on page 139 for further explanation of the options for this utility.

Archive Logging

The **osarchiv** utility takes much smaller pictures of the database than **osbackup**. Object Design 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. Since 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 completes successfully. Avoid updates in the window of time following the start of **osbackup** and the startup of **osarchiv** so **osarchiv** does not have to dump full segments. 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 megabytes. 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:

YYMMDDHH.ext, where,

YY =	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. 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. (Can this add to the recovery process?) However, since 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 where 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.txt -d archive_img -i 30 -C *sample1.db sample2.db sample3.db*

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

- The **-a** flag specifies the pathname of the 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 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 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 automatically taken. You can take a snapshot at any time that **osarchiv** is active by issuing the **x** command. Note that the **x** command can only be used when running **osarchiv** interactively.
- The **-C** flag enables the interactive command-loop feature. This feature is disabled by default.

Running **osarchiv** interactively

While the **osarchiv** utility is running with the interactive mode enabled, you can execute the following commands. The utility processes the command between snapshots.

a *pathname*	Adds the specified file database or rawfs database or directory to the archive set.
h	Displays on-line help.
i *interval*	Changes the interval between snapshots. Specify an integer for *interval*.
n	Closes the current archive file and starts saving snapshots in the next archive file in the sequence.
q	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	Displays the pathnames of the databases and rawfs directories in the archive set.
X	Takes a snapshot as soon as you issue the command. This has no effect on snapshot intervals.

See osarchiv: Logging Transactions Between Backups on page 132 for further information about the options for this utility.

Special Considerations for Large Databases

The **osbackup** procedure backs up approximately 4 gigabytes 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 20 gigabytes an hour. If you have very large databases to back up, the advantage of an on-line back up with **osbackup** might be less important than the amount of time it takes to do the back up. You might benefit more from shutting the ObjectStore system down and backing up the file system using operating system commands, as described next.

Using a file system backup and the ObjectStore archiver

If you keep the ObjectStore's archiver running while the operating system backup is running, the database is never without a recovery mechanism. 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 Freeze your applications (or shut them down).

2 Perform a Server checkpoint.

3 Wait for the archiver to take its snapshot.

4 Shut down the archiver.

5 Shut down the Server.

6 Run the File system backup.

7 Restart the Server.

8 Restart the archiver.

9 Resume or restart your applications.

Steps 3 through 5, 7, and 8 are only necessary if clients might access the Server during this procedure. If you can guarantee no client access, then you need only perform steps 1, 2, 6, and 9.

The next paragraphs describe how **osrestore** and **osrecovr** work.

Recovery Options

You should use the **osrestore** utility to restore databases from the most recent backup images created previously by **osbackup**. Recover any archived 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 then 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\97080719.aaa
  archive_img\97080719.aab
  archive_img\97080720.aac
  archive_img\97080720.aad
  archive_img\97080720.aae
  archive_img\97080720.aaf
  ```

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

-F archive_list.txt -D 7/8/97 -r 19:59:45

See osrestore: Restoring Databases from Backups on page 216 and osrecovr: Restoring Databases from Archive Logs on page 206 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, then invoke **osrecovr** with the file name that contains a list of all the archived images in the directory that was specified by the **-d** flag.

The Dump/Load Subsystem

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

The ObjectStore dump/load facility allows you to

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

The dumped ASCII file is in a compact, human-readable format. You can edit the file using an editor such as Perl, Awk, or Sed. The ASCII file (edited or unedited) is the input for the loader.

See osdump: Dumping Databases on page 177 and osload: Loading Databases on page 196 for more detailed information about using the dump/load facility. For advanced topics related to customization, see Chapter 8, Dump/Load Facility, in the *ObjectStore Advanced C++ API User Guide*.

Managing Users

There are two sorts of ObjectStore users:

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

This explanation focuses on what is required for those running ObjectStore applications.

Running applications

To run an ObjectStore application, you need

- Client, Server, and Cache Manager executables.
- 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 **OS_ROOTDIR**. **OS_ROOTDIR** is an environment variable that 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 you build your application. The application schema database contains the definitions for all classes the application uses in a persistent context. An ObjectStore executable embeds the pathname of the application schema database. Therefore, you must ensure that the application schema database is available to the executable.

You can use the **ossetasp** utility to change the pathname of an application schema database for a particular executable, except on OS/2.

Alternatively, an application can specify its application schema database pathname 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 exactly matches the schema for an executable. ObjectStore compares the application schema with the database schema during execution.

Developing applications

To develop an ObjectStore C++ application, you need to be able to run **ossg**, the ObjectStore schema generator. Developers use the schema generator to create the application schema database. See *ObjectStore 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. One reason you might want to do this is if you are running two releases of ObjectStore on a machine at one time. Another reason to modify port numbers might be if your site uses a firewall to prevent unauthorized network access. You can choose to have the firewall protect the default port numbers shown below or you can select different port numbers that are protected.

Communication Methods Background

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 below. 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.

The following table lists the communication channels for each platform.

Platform	*Local Transport Layer*	*Remote Network*
OS/2	**Named Pipes**	IP
AIX Digital UNIX HP–UX	**UNIX** (uses UNIX domain sockets)	IP
SGI Solaris 2	**TLI_LOCAL**	IP
Windows	**NSharedMemory**	IP

UNIX

On UNIX, the remote network is always TCP/IP. For the local network, ObjectStore uses sockets to communicate among Server, client, and Cache Manager processes. Sockets use predefined ports that you can modify if necessary. A port identifies a

particular application on that machine. (An address identifies a particular machine.)

Windows and OS/2

On Windows and OS/2, ObjectStore uses a variety of remote networks. Each mechanism has something analogous to port addresses that you control with the ports file.

Defaults for Port Settings

For **Named Pipes**, NetBIOS, and **NSharedMemory**, the port setting modifies the default port, so you can use the same setting for all services. On the other networks, a port setting replaces the existing port setting, so you must provide unique settings for each port that you modify.

All Servers in all ObjectStore versions have the same port number. The default port settings are in the following table.

Process	*IP Port Number*	*TLI_LOCAL Pathname*	*UNIX Pathname*
Server to R4 Cache Manager	**51034**	**os_callback_v4**	**/tmp/ostore/os_callback_v4**
R4 client to R4 Cache Manager	**51044**	**osccom4**	**/tmp/ostore/osccom4**
Client to Server	**51025**	**objectstore_server_comm**	**/tmp/ostore/objectstore_server_comm**
Notification	**51049**	**objectstore_notification**	**/tmp/ostore/objectstore_notification**

Modifying Port Settings

To modify the settings, you change entries in the 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 the ports file.

On UNIX, Windows, and OS/2, 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 and OS/2	**%OS_ROOTDIR%\ETC\PORTS**

Ports File 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 appear 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.

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 Release 4, specify **4**. For compatibility, ObjectStore Release 5 also requires **4** as the version number in all cases except for **server client**. This service requires a **1**.

port Specifies a port identifier as described in the next table.

Network	*Port Identifier*	*Port Identifier Example*
IP (TCP/IP)	TCP/IP 16-bit port number. See Defaults for Port Settings on page 52.	**51025**
Named Pipes	Second element of pathname. The default is **ostore**. The format is **\\PIPE***second***\oss_server_client**	**\\PIPE\ostore\oss_server_client**
NetBIOS	Last character of string you specify. The default is the capital letter **O**. The format is *name_of_your_choiceX*	**YOUR_HOST_NAMEO**
NSharedMemory	First character of shared object name. Omitted by default. The format is *X***accept.oss_server_client**	*X***accept.oss_server_client**
TLI_LOCAL	Pathname in TLI Local name space. No default.	**objectstore_server_com**
UNIX	Pathname in file system. No default.	**/tmp/ostore/objectstore_server_com**

Example **TCP/IP:server client:1:54432**

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

UNIX domain example

```
UNIX:server client:1:/tmp/.ostore/objectstore_server_comm
UNIX:cache manager client:4:/tmp/.ostore/objectstore_client_cache
UNIX:cache manager server:4:/tmp/.ostore/objectstore_server_
cache
```

Running Two Servers on One Host

You can run two ObjectStore Servers on the same host under some circumstances. You must specify different ports in the ports file. This lets clients know which one to contact.

Different versions

Object Design recommends that you never run two Servers that are the same ObjectStore version on the same host. If you want to run two Servers on the same host, ensure that they are different ObjectStore major versions, for example, one release 4 and one release 5 version.

To set up two Servers on the same host, you only need to change the **server client** ports for one of the Servers. The Cache Manager ports are already different in different major versions of ObjectStore.

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 and SGI, specify a combination of **TLI_LOCAL** and **TCP/IP** statements. Note that the **TLI_LOCAL** domain port names exist in a separate, flat name space.

On non-System V Release 4 systems, such as AIX, Digital UNIX, and HP–UX, specify a combination of **UNIX** and **TCP/IP** statements in the ports file. The UNIX domain or local ports have names in the file system's name space.

Windows

Object Design does not support running two Servers on one host. On Windows systems, you can specify statements for NSharedMemory, TCP/IP, and NetBIOS, according to which networks you use.

OS/2

On OS/2 systems, you can specify statements for Named Pipes, TCP/IP, and NetBIOS, depending on which networks are in use.

If you do not specify port settings correctly, and you try to start a second Server or Cache Manager, the operation fails with the following error:

<err-0001-0144>There is a server for the service <sss> already running on net <nnn>.

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 the receipt of) incoming notifications from the Cache Manager. (Outgoing notifications are sent to the Server, not 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 what network port is used by the connection. You can use the ports file in the usual way to control the use of network ports. The relevant information follows.

Name of service	**cache manager notification**
Default IP port number	**51049**
Default UNIX local socket file name	**/tmp/ostore/objectstore_notification**

How a Client Locates the Server for a Database

How does a client determine which 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 pathname for the database. 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 281.

If there is a locator file, the client checks it to determine which 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 one of two ways:

- The rawfs host name appears in the pathname of the rawfs database.
- The environment variable **OS_DIRMAN_HOST** is set to the name of the rawfs host.

See Rawfs Databases on page 5.

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 only one 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, close) work on the file, ObjectStore can successfully use them.

Translating Pathnames

You might run an ObjectStore application on both PCs and UNIX systems. UNIX uses forward slashes in pathnames. PCs use backward slashes. How does ObjectStore handle this?

Suppose you have a UNIX Server and some Windows NT and OS/2 clients.

Native pathnames

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 where 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**.

Server-relative pathnames

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

Callback Messages Background

The Cache Manager facilitates concurrent access to data by handling callback messages from the Server to client applications. To understand how callback messages work, you need to know about *ownership* and *locks*.

Note: The terms *ownership*, *encached page*, and *permit* all represent the same concept — a client has permission to read or modify the page.

Read/write ownership

Read ownership exists when a client has permission to read a particular page. Write ownership exists when a client has permission to modify a particular page.

A Server grants read ownership to as many clients as request it. If one client requests write ownership, that client must wait if there are current transactions that are reading that page, or if there is a current transaction that is modifying that page.

Many clients can have read ownership at the same time. Only one client at a time can have write ownership. When a client has write ownership, no clients can have read ownership.

Clients can release ownership in the following situations:

- The result of a callback message to the Cache Manager is that the client can release ownership.
- The client terminates its connection with the Server.

Read/write locks

When a client actually reads or writes a page during a transaction, it places a read or write lock on that page. When a client has a lock on a page, it means that another client cannot receive write permission for that page. A client must have read or write ownership to be able to place a read or write lock on a page. The client releases the lock when the transaction commits.

Ownership and locks

It is important to recognize the difference between ownership and locks. Ownership gives the client permission to read or modify a page. A lock allows the client to actually read or modify the page.

When a client has ownership, it can place a lock on a page without communicating with the Server.

A client can have read ownership without a read lock if it previously read the page but is not reading the page in the current transaction. The same is true for write ownership and locks.

Lazy release

The client does not give up ownership on pages when the pages are no longer needed. The client keeps ownership in case data on those pages is needed again. In contrast, the client always gives up locks when a transaction commits.

The lazy release is an optimization. If no other client needs the page that your client has ownership of, then your client does not need to request ownership from the Server for that page for as long as the client is active.

Callbacks

When a client requests read ownership for a page and no other client has write ownership, the Server grants read ownership. The Cache Manager is not involved.

The Cache Manager is involved in the following situations:

- When a client requests read or write ownership for a page and another client already has write ownership for that page
- When a client requests write ownership for a page and other clients have read or write ownership for that page

In these situations, the Server sends a callback message to the Cache Manager on the host of the client that has ownership. The Cache Manager determines whether ownership can be released or the client requesting ownership must wait.

If the client with ownership

- Has a lock on the page because it is in a transaction, the second client waits.
- Does not have a lock on the page, perhaps because it is between transactions, the second client receives ownership.

  If the first client had write ownership, and the second client is requesting read ownership, the Cache Manager downgrades the ownership of the first client from write to read. Otherwise, the client loses ownership.

API for lock management

Normally, ObjectStore performs lock management. There are, however, API features that allow a client to control whether or not a client waits for a lock. When an application uses these features, ObjectStore still uses the lazy release.

For more information about ownership and locks, see Locking in Chapter 3, Transactions, of the *ObjectStore C++ API User Guide*.

Troubleshooting

This section provides some 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.

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

Server Initialization Failed

You might try to start the Server and receive this message:

```
Server initialization failed
Trouble accessing ObjectStore file system.
Exception message: <maint-0018-0006>
File access permission denied
```

The Server is not starting. The resource affected is the Server. To obtain more information, you can

- Check the content of the Server output file.

OS/2	**OSSERVER.TXT**
UNIX	**/tmp/ostore/oss_out**
Windows	**%OS_ROOTDIR%\OSSERVER.TXT**

- Ensure that you are starting the Server with the correct permissions. For example, on UNIX you must be **root**.
- On UNIX and OS/2, you can try to restart the Server in debug mode.
- Consider what the Server needs to do when it starts up. See What Does the Server Do? on page 7.

Here is another message you might receive when you try to start the Server:

```
Server initialization failed. (orecvery.cc line 2787)
Unknown record type 127528668 at line 1266304
Failed to start ObjectStore server
```

Again, the Server is not starting. The resource affected is the Server and the reference to *record type* might indicate a problem with the log. To obtain more information, you can

- Check the content of the Server output file. For the name of the file on your platform, see the table in **OSSERVER.TXT** on page 61.
- 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 the log might not have the correct permissions.

Miscellaneous ObjectStore Error

While running a client application you might receive this message:

```
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

- Check the contents of the Cache Manager output file to see if the Cache Manager is running.

OS/2	**OSCMGR4.TXT**
UNIX	**/tmp/osc4_out**
Windows	**OSCMGR4.TXT**

- Check cache and commseg file locations and dates. You might have deleted these files, but did not use the **oscmrf** utility.

UNIX

- On UNIX, verify **suid root** permissions on the **oscminit** executable.

  If you are mounting the file system that contains the **oscminit** executable, be sure that you do not specify **-o nosuid**. When you mount the file system, **-o suid** is the default; you do not need to

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.

No Handler for Exception — Database Error

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

No handler for exception:
ObjectStore internal error Database *dbname* has fraction length?
(err_internal)

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

- Use the **osverifydb** (see osverifydb: Verifying Pointers and References in a Database on page 274) and **ossize** (see ossize: Displaying Database Size on page 248) utilities to check all databases that the application accessed. Check both schema and production databases.
- Use a debugger to find the line of code causing the error.
- Confirm that the architecture page size matches the database page size. This error can be caused when an application running on a machine that uses 8K pages tries to access a database created on a machine that uses 4K pages.
- Rebuild the database, checking for corruption as it occurs.

No Handler for Exception — Cannot Open Application Schema

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

No handler for exception:
Cannot open the application schema
<err-0025-0311> The application schema database db.asdb 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

- Check that a current application schema database exists. It might be missing.
- Use the **ossetasp** utility (see ossetasp: Patching Executable with Application Schema Pathname on page 229) to check that the

executable has the correct location of the application schema database. The executable might have the wrong location.

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 281.

No Handler for Exception — Invalid Address

When trying to run a client application you might receive a message like this:

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

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

Unsupported Server Protocol

If you are running a Release 5 Server and a non-Release 5 Server on the same machine, you might receive the following message when trying to run an application.

```
Attempt to use unsupported server protocol.
<err-0001-0006> Server elvis does not support version 4.0 clients.
(err_protocol_not_supported)
```

ObjectStore displays this message when a Release 5 client tries to communicate with a non-Release 5 Server. This error might occur if you did not shut down the non-Release 5 Server while you

installed the Release 5 Server. Shutting down and restarting the non-Release 5 Server should correct the situation.

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 usually 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.

AIX — Available Space But Database Does Not Grow

Your ObjectStore database does not grow past 1.073 gigabytes, even though you have available disk space. You receive the following error:

```
No handler for exception:
The server is out of disk space for the database or logfile
set_segment_size call to server (host "rexx")
```

When you check your system, you might have statistics such as the following

```
rexx:phawkins(678)> df .:
Filesystem    Total KB free  %used   iused  %iused  Mounted on
/dev/lv00     2002944        411296  79%    5439    1% h/rexx/2

rexx:phawkins(679)> ls -l mydb
-rw-rw-r--   1 phawkins sys      1073082368 Sep 26 17:26 mydb
```

The problem is that you are reaching the AIX operating system default limit for the **root** process. This limit is 2097151 blocks, or 1073741312 bytes. The exact database size that the failure occurs at varies.

To change the limit, change the file **/etc/security/limits** and add **-1** for the **root fsize** limit. For example:

```
default:
    fsize = 2097151
    core = 2048
    cpu = -1
    data = 262144
    rss = 65536
    stack = 65536
root:
    fsize = -1
daemon:
bin:
sys:
adm:
uucp:
guest:
```

Before You Call Object Design Support

When you contact Object Design Technical Support, you need to provide

- Hardware platform
- Operating system and release number
- ObjectStore release number
- Severity of the problem
- Description of the problem
- Information found in the Server and / 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

Using Virtual File Systems

A virtual file system provides a user with a logical view of a file system. When you use a virtual file system, it appears as if all sources and executables reside in the current directory path, when only local modifications actually reside there. The purpose of a virtual file system is to hide the actual locations of files from users. ClearCase is an example of a virtual file system.

Opening databases

You can run ObjectStore clients and utilities in directories where ObjectStore can find all files. However, the opening of a database is always performed by the Server. In a virtual file system, the Server cannot determine where a file actually resides unless you specify the true location.

Pathnames you can use

When you use ObjectStore with a virtual file system you must specify pathnames that would work without the virtual file system. This is usually some kind of absolute pathname.

This requirement applies to any ObjectStore database stored under a virtual file system and includes any object that the Server must access. The Server must be able to access whatever pathname is provided. This can be a virtual file system pathname if the Server is running under a virtual file system.

Running Server under virtual file system

You can run the Server under a virtual file system if there is only one *view* of the virtual file system and the view is shared by all users. The Server cannot recognize more than one view. In some virtual file systems, machines and users can have their own views.

Schema databases

You can maintain schema databases in a virtual file system because the schema generator embeds the absolute pathname of the schema in the database. However, when you generate schema databases, you must specify pathnames that the Server can use.

Some virtual file systems provide a way to return an absolute (nonvirtual) pathname for a given element in a virtual file system. This pathname can enable the ObjectStore Server to locate the element. It does mean the loss of the ability to use leaf names (such as **eng_1.libschema** in the following example) as arguments to ObjectStore utilities.

Sample error message

Here is an example of the type of message you might receive when ObjectStore cannot find a file. Notice that the file appears to be there.

```
> pwd
/home/clients/libschemas
> ls
test_1.libschema
eng_1.libschema
qa_1.libschema
> ossize eng_1.libschema
ossize: The database was not found
<err-0025-0351>The database
"/home/clients/libschemas/eng_1.libschema"
does not exist (err_database_not_found)
```

Chapter 2
Server Parameters

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

To modify a parameter, seem the instructions in the chapter for your operating system. After you modify a Server parameter you must shut down and restart the Server for the change to take effect.

Parameters are available on all platforms unless otherwise noted. Case is not significant. Defaults are in the left margin.

The parameters described in this chapter are

Admin Host List

Default: none

The **Admin Host List** parameter specifies the pathname 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.

When you specify a host in the file pointed to by the **Admin Host List** parameter, it causes that host to require only **SYS**

authentication. This is true regardless of that host's setting of the **Authentication Required** Server parameter.

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, you 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 for 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

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 / write, the Server holds a write lock.

When an ObjectStore Server holds a read lock on a database, this prevents other Servers from acquiring a write lock on the database. Read access by other 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, this prevents other Servers 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 automatically retried.

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 and OS/2

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 that Server has a value of **yes** for this parameter, the Server handles access to the database. If the Server does not have a value of **yes**, 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 / write. This is because when one Server handles access to a database, it can prevent concurrent access by other ObjectStore Servers. See the Server parameter **Allow NFS Locks**.

Caution

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

Allow Shared Communications

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. Doing so improves performance. It is a Boolean that defaults to **yes** (meaning shared memory communication is enabled). If these communications are enabled, the Server and client exchange data by means of shared 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. That's 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

Default: by platform

An understanding of ObjectStore authentication is necessary before an explanation of the **Authentication Required** parameter makes sense.

The defaults are

- **NONE** on OS/2 and Windows 95
- **Name Password** on Windows NT
- **SYS** on other platforms

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 checking 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 does 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.

Parameter values

The **Authentication Required** Server parameter specifies how the Server controls database access. There are five possible types of authentication:

- **NONE —** The only available option on platforms that do not support security (OS/2 and Windows 95).

On UNIX platforms, clients who present authentication **NONE** cannot access file databases. Release 5 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 that is accessible to the user running the Server (usually Local System).

- **SYS —** UNIX only. ObjectStore uses the Sun ONC RPC **AUTH_SYS** authentication method, previously called **AUTH_UNIX**.

  The client sends the Server a UNIX user ID and a set of group IDs, which the Server trusts. The ObjectStore client library always sends the current effective user ID and group set. This is the same mechanism used by the NFS protocol.

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.

- **DES —** Sun, AIX, 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.

- **Name Password** — All platforms except OS/2 and Windows 95. This value was previously named **UNIX Login**. In this release, **UNIX Login** is a synonym for **Name Password**.

  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 78. 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 then retain passwords.

- **NT Local** — Windows NT only. Allows Windows NT clients to communicate with local Servers.

Parameter value might imply a set of values

When you set the parameter to one of these authentication types, it means that 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. The complete prioritized list of authentication types is

1 **NONE**

2 **SYS**

3 **DES**

4 **NAME PASSWORD**

5 **NT LOCAL**

To determine the list of authentication types that a Server supports, run the **ossvrstat** utility. See ossvrstat: Displaying Server and Client Information on page 261for further information.

Examples of allowable authentication

For example, if an AIX Server has the value **NONE** for the **Authentication Required** Server parameter, it means that a client can return information that complies with **NONE**, **SYS**, **DES**, or **NAME PASSWORD**. These four values make up the Server-supported list on this platform.

If an HP Server has **Authentication Required** set to **SYS**, it means that a client can return information that complies with **SYS** or **NAME PASSWORD**. The Server-supported list for HP–UX has two values.

If a Windows NT Server has **Authentication Required** set to **Name Password**, clients can return information that complies with **Name Password** or **NT Local**. The Server-supported list for Windows NT contains two values.

How does a client choose?

How does a client determine which type of authentication to provide to the Server?

On all platforms, if **Authentication Required** is **NONE** and if the Server can have a value of **SYS** for the parameter, the client returns information that complies with **SYS**. This allows Release 5 clients to access Release 5 rawfs and file databases.

Suppose that **Authentication Required** is **NONE** but the Server cannot have a value of **SYS**. In this case, the client returns information that complies with the first value in the Server-supported list that the client supports. For an AIX client contacting a Windows NT Server, this would be **Name Password**. Note that if the Server were on AIX, both the **SYS** and **DES** values would be allowed. A Server that cannot have a value of **SYS** is not a UNIX system and so the Server-supported list of values for authentication includes **NONE**, **Name Password**, and possibly **NT Local**.

UNIX clients always provide **Name Password** authentication when accessing Windows NT Servers.

Windows NT clients

When a Windows NT client provides authentication it must provide the domain name, the user name, and the user password. By default, ObjectStore uses the **USERDOMAIN** environment variable. You can override this by specifying a name of the form *domain******name*.

When a Release 5 Windows NT client contacts a local Windows NT Server, the client always provides **NT Local** authentication.

When a Release 5 Windows NT client contacts a remote Windows NT Server or a UNIX Server, the default is for the client to provide **Name Password** authentication.

To allow a Windows NT client to provide **SYS** authentication to a UNIX Server, you must set the user ID and group ID 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. See Chapter 8, Managing ObjectStore on Windows, on page 345.

Suppose a Release 5 Windows NT client contacts a 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. This allows compatibility with UNIX Servers. If the Server does not allow **SYS** authentication or if the user ID and group are not set in the registry, then the client provides the first type of authentication that it can that is in the list of Server-supported authentication types. In some cases, this is **Name Password**. But it might be **NONE**, which means that a Release 5 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 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, it 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 *ObjectStore C++ API Reference*.

Cache Manager Ping Time

Default: 300

The **Cache Manager Ping Time** parameter specifies a number of seconds. Whenever between **Cache Manager Ping Time** seconds and twice **Cache Manager Ping Time** seconds have elapsed with no message from the client, the Server attempts to send a message to

the client's Cache Manager to check 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 that is 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

The **Cache Manager Ping Time In Transaction** parameter is the same as the **Cache Manager Ping Time** parameter, except 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.

DB Expiration Time

Default: 300

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 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 which client is the victim. The

Server sends a message to the selected client to abort the transaction.

Age	Youngest client. A client's age is calculated as the time since its last successfully committed transaction.
Current	Client that made the last request to the Server, causing it to detect the deadlock.
Oldest	Oldest client. A client's age is calculated as the time since its last successfully committed transaction
Random	Random client.
Work	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 *ObjectStore C++ API Reference* for more information on transaction priorities.

Deadlock

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.

Transaction data

When deadlock occurs, the Server automatically detects it and chooses a victim whose transaction is aborted. If the aborted transaction is a lexical transaction, it is automatically retried. 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

The **Direct to Segment Threshold** parameter specifies, in sectors, the threshold value that determines whether the Server writes segments to the log first and then to the database, or writes them directly to the database. If less than this threshold is written past the current end of 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. The default is **128** sectors (64 KB).

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 8 to 12 milliseconds per access, it would take 800 milliseconds to 1.2 seconds to perform all the write operations. But if 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 only take 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 how 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 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 last sector number 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.

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 advent of a failure, it takes five times **Failover Heartbeat Time** for the secondary Server to recognize the failure and to take over.

Host Access List

Default: none

The **Host Access List** parameter specifies the pathname 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, such as NetBIOS, there might not be a secure method.

This parameter is intended for use in environments where a machine is on a network with untrustworthy hosts and **DES** authentication is 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.

Log Data Segment Growth Increment

Default: 2048

The **Log Data Segment Growth Increment** parameter specifies how many sectors to add to the data segment in the transaction log when more room is needed. The default is **2048** sectors (1 MB).

Log Data Segment Initial Size

Default: 2048

The **Log Data Segment Initial Size** parameter sets the initial size of the data segment in the transaction log in sectors. The default is **2048** sectors (1 MB).

Log File

Default: rawfs

The **Log File** parameter specifies the pathname 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 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 should not specify a name for this parameter. See Description of the Server Transaction Log on page 17.

If you specify a pathname for this parameter, the pathname 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 accidentally deleted by a user. You might want to name it in such a way that it is never deleted.

Caution: Deleting the log file can cause database corruption.

Windows and OS/2

On Windows and OS/2, 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. The defaults are in the following table.

OS/2	**OSSVXOS2.LOG**
Windows NT	**OSSVXNT.LOG**

Log Record Segment Buffer Size

Default: 1024

The **Log Record Segment Buffer Size** parameter defines how much buffer space is reserved for log records. The default is **1024** sectors (512 KB).

Log Record Segment Growth Increment

Default: 512

The **Log Record Segment Growth Increment** parameter specifies by how many sectors to increase the log record segment when more room is needed. The default is **512** sectors (256 KB).

Log Record Segment Initial Size

Default: 1024

The **Log Record Segment Initial Size** parameter sets the initial combined size of the two log record segments in sectors. The default is **1024** sectors (512 KB).

Max AIO Threads

Default: 3

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.

When this parameter is set to **0**, Servers perform I/O synchronously.

If you never open more than a given number of files, you might want to set this parameter to that number. This ensures 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.

The default setting is **3** threads.

Max Connect Memory Usage

Default: 0

The **Max Connect Memory Usage** parameter defines in kilobytes an amount of virtual memory. When the Server is using this much virtual memory, it will refuse 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 how much 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.

The default is that **Max Connect Memory Usage** is unlimited. This is indicated with a **0**.

Max Data Propagation Per Propagate

Default: 512

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. The default is **512** sectors (256 KB).

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 its default of **512** sectors, a maximum of eight noncontiguous writes can occur in a single propagation.

Max Data Propagation Threshold

Default: 8192

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 the value specified for **Max Data Propagation Threshold** the Server forces propagation to run faster. The default is **8192** sectors (4096 KB).

Max Memory Usage

Default: 0

The **Max Memory Usage** parameter specifies in kilobytes the maximum amount of virtual memory the Server can use. The Server checks this parameter when it starts. The Server immediately allocates and then frees the specified amount of memory. On some platforms, this 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 (about 64 KB per client)
- Amount configured for message, propagation, and log buffers

The default is that **Max Memory Usage** is unlimited. This is indicated with a **0**.

Max Two Phase Delay

Default: 30

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

The **Message Buffer Size** parameter sets the size of the message buffer that the Server uses for processing a message from a client. The Server reads data from this buffer and writes data requested by the clients into this buffer. The default is **512** sectors (256 KB), with the value being rounded to a multiple of 64 KB.

Message Buffers

Default: 4

The **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

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: 16,384

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: 8192

The **Propagation Buffer Size** parameter selects the amount of buffer space reserved for propagation. This space is used for reading data from the log that is 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 occurs, thus avoiding the need to read the log. The default is **8192** sectors (4096 MB).

Propagation Sleep Time

Default: 60

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 temporarily decreases this interval.

Restricted File DB Access

Default: none

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, then

- ObjectStore does not allow access to rawfs databases.
- 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 does not allow access to file databases.

Chapter 3
Environment Variables

This chapter describes the ObjectStore client environment variables.

These alphabetically-ordered variables are described in the pages that follow:

Specifying Values for Environment Variables

You can set environment variables to modify the characteristics of the client 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.

An environment variable is available on all platforms unless noted otherwise in this chapter. Defaults appear in the left margin.

The value for a variable is either an integer, a Boolean, 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 (nonempty), ObjectStore interprets it as **0** or false.

Specifying integers

When a 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 Booleans

When a 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 / 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**

While the following values return **false**:

- **0**

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

OS_AS_SIZE

Default: by platform

Sets the size, in bytes, of the persistent storage region in the address space for each client. If you do not specify a value for this variable, which is common, the default depends on your platform, as shown in the table under the heading Platform on page 95. Specify numbers of bytes in decimal values.

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 segment is smaller than **OS_AS_SIZE**, then the largest object you can store is limited to the maximum segment size. (Allocation is limited to a single segment.)

Caution

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

For example, if you quadruple **OS_AS_SIZE**, your application runs on a SPARCserver / 470, but fails on SPARC 1 and SPARC PC systems.

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 Server determines where it would put those pages if the client needed them. Assigning an address reserves it so that it cannot be assigned to another page. When ObjectStore maps a page to an address, it means that the page is available to the client.

ObjectStore assigns address space to any pages outside the segment containing the initial page that was referred to. 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 where the schema design could benefit from the strategic application of references, and sometimes a reworking of transaction semantics.

If the application uses a reference, instead of a pointer, to point to data in other segments, ObjectStore does not assign address space to the other segment until the reference is actually resolved. In programs where a complex schema requires the use of a segment that contains pointers to lots of data in other segments or where an extent contains pointers to instances in many segments, use references instead of 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.

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

OS_AS_START

Default: by platform

Sets the address of the beginning of the persistent storage region of the process's address space. If you do not supply a value for this variable (this is the usual case), the default depends on the platform, as shown in the table under the heading Platform on page 95.

Individual platforms impose further constraints on what values are legal.

See **OS_AS_SIZE** on page 93 for a caution about the use of this variable.

If **OS_AS_START** and **OS_AS_SIZE** are the same on all machines that create and use data in a given database, ObjectStore can often optimize relocation, which improves performance. See **OS_RELOPT_THRESH** on page 117 for a discussion of the relocation optimization.

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

The following table shows the specifications for the persistent storage region in a process's address space.

Platform	*Default Starting Address* **OS_AS_START**	*Default Size* **OS_AS_SIZE**	*Default Ending Address*
Digital UNIX	0x35000000	512 MB+ (0x2003E000)	0x5503E000
HP–UX	0x563A0000	300 MB (0x12C08000)	0x68FA8000
IBM RS/6000	0x40000000	1024 MB (0x40000000)	0x80000000
IBM OS/2	0x01000000	128 MB (0x08000000)	0x09000000
Intel Windows NT and 95	0x30000000	128 MB (0x08000000)	0x38000000
SGI MIPS	0x30000000	255 MB (0x1900000 0FFF0000)	0x3FFF0000
Sun SPARC Solaris 2	0xE2000000	192 MB (0x0C000000)	0xEE000000

Windows NT

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

SPARCstation 10

There is no address space hole on the SPARCstation 10. In a single **mmap**, you can map any portion of the address space from address 0 to the process stack (around 0xefffe000). However, a single call to **mmap** has a limit of 0x80000000 bytes (approximately 2 GB). For ObjectStore, that means you can only use half of the entire 32-bit address space.

SPARCstation 1000

On a SPARCstation 1000, the address space available for mapping starts above the *break*. The break is the top of the process's heap. The heap grows by increasing the break. It is risky to map a persistent range that is too near the break because a growing heap might collide with the persistent range.

The hole in the SPARCstation 2's address space begins at 0x20000000 and ends at 0xe0000000. Addresses above the hole are

available for mapping, but ObjectStore requires that the persistent range be contiguous, so the range cannot span the hole.

On Solaris 2, the default start address is 0xe2000000, just above the end of the hole. The persistent range can be placed on either side of the hole on both systems.

On a SPARCstation 2 running Solaris 2, the size of the persistent range is limited on the high end by the area where shared libraries and other objects placed by the system are allocated. This starts below the stack, around 0xf0000000, and grows downward.

On the SPARCstation 1000, though, this area starts around 0xe0000000, and grows downward. Apparently, on the SPARCstation 1000, ObjectStore's default range is sandwiched between the stack and shared libraries. Since the hole does not constrain the available space on this machine, the system gives the stack a bit more room to grow by placing shared libraries at lower addresses.

For a maximum persistent range on the SPARCstation 1000, the range from 0x10000000 to 0xd0000000 should be safe to use and large enough for just about any purpose. However, you should test this before you put an application into production that depends on it.

SPARCcenter 2000

Up to 1.8 gigabytes can be allocated in a single **mmap** call.

OS_AUTH

Overrides NT Remote authentication. The following table lists values that **OS_AUTH** can be assigned:

Value	*Authentication Override*
0	Client sends no authentication
1	Client sends SYS authentication
3	Client sends DES authentication
10	Client sends Name Password authentication
12	Client sends NT Local authentication
13	Client sends NT Remote authentication

OS_BOOTSTRAP_LRU_CACHE_SIZE

Default: three-quarters of cache size

Specifies a number of bytes. When the cache grows to this size, ObjectStore performs an LRU (least recently used) bootstrap routine to start the process of determining which pages are not being used. This allows the first page eviction to be done intelligently, since page eviction chooses pages to evict based on information that must be collected over time. If the size of your cache drops below the **OS_BOOTSTRAP_LRU_CACHE_SIZE** threshold, ObjectStore suspends this process until you reach the threshold again. This way your application does not have the unnecessary overhead of determining which pages should be evicted.

OS_BOOTSTRAP_LRU_CACHE_SIZE has a default setting that is three-quarters of the cache size.

If your cache is large enough to hold the entire database being used by a program, set **OS_BOOTSTRAP_LRU_CACHE_SIZE** so that it is greater than the cache size. This improves performance by avoiding execution of the bootstrap routine. Execution is unnecessary since page evictions are not needed when the cache can hold the entire database.

When you set **OS_BOOTSTRAP_LRU_CACHE_SIZE** to a size greater than (**OS_CACHE_SIZE** – *eviction_pool_size*), ObjectStore runs the LRU bootstrap routine when there are *eviction_pool_size* free pages left in the cache. The *eviction_pool_size* defaults to 2 percent of the cache and is always at least twice the *eviction_batch_size*, which defaults to 1 percent of the cache.

Use the API to set eviction sizes: **objectstore::set_eviction_pool_size** and **objectstore::set_eviction_batch_size.** See *ObjectStore C++ API Reference*.

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

OS_BROWSER_NUMERIC_FORMAT

Default: ObjectStore defaults

Allows you to set the output format of numeric types. It supports the following types: **int**, **long**, **double**, **ldouble**, **float**, **uint**, **ulong**, **short**, and **ushort**. If you do not set any of these types, ObjectStore defaults are used.

The following example sets the display format of all **int** types to hex, all **float** types to four-digit precision, and all **ulong** types to hex.

```
OS_BROWSER_NUMERIC_
FORMAT="int=%x,float=%.4f,ulong=%x"
```

OS_CACHE_DIR

UNIX

Default: **/tmp/ostore**

Specifies the pathname of the directory in which ObjectStore places the client cache and communications segment (commseg) files. Specifying an alternative pathname can be useful when your **/tmp/ostore** directory is small. ObjectStore places the cache file 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 333.

This directory should not be an NFS mount point because this can result in slower client performance and can result in potential problems with memory mapping over NFS.

Cache and commseg files should be in the same directory. See **OS_COMMSEG_DIR** on page 101 for related information.

Windows and OS/2

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

OS_CACHE_SIZE

Default: 8 MB
ObjectStore Single: 2 MB

Size of client cache, in bytes. The cache size defaults to 8 MB, except for ObjectStore single applications, for which the default cache size is 2 MB. In either case, 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 trying to determine the optimum cache size for your application, consider data access patterns as well as how much

data 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.

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

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

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

OS_CMGR_STARTUP_LOCK

Default: varies by platform

Specifies an alternate location for the Cache Manager start-up lock file.

On UNIX and OS/2 systems, ObjectStore clients automatically start a Cache Manager if one is not already running. ObjectStore uses a special file, called a start-up lock file, to ensure that only one Cache Manager starts on a host. The default location for this file is

OS/2	**\SEM32\OSTORE\CMGR4_STARTUP_L**
Solaris 2	**/var/tmp/cmgr4_startup_lock**
Other systems (not Windows)	**/tmp/cmgr4_startup_lock**

If a Cache Manager is not already running, a client

1 Creates the start-up lock file

2 Launches the Cache Manager

3 Ensures that it can communicate with the Cache Manager

4 Deletes the start-up lock file

The start-up lock file does not contain anything. Its existence serves as a lock while a client is starting a Cache Manager. This prevents other clients from also starting a Cache Manager.

If you do not want to use the default location for the start-up 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 pathname.

ObjectStore tries to use the default first. If it is not possible to use the default, ObjectStore uses the pathname you specified for **OS_CMGR_STARTUP_LOCK**.

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

OS_COLL_POOL_ALLOC_CHLIST_BLKS

Default: false

Specifies that chained list blocks should be pool allocated. To set this variable, specify any nonnull value except **0**.

OS_COLL_THREAD_LOCKS

Default: true

Determines whether ObjectStore uses a lock to ensure that only one thread at a time can execute within ObjectStore collections code.

When this variable is set to a nonnull value other than **0**, ObjectStore uses the collections thread lock. This is the default.

If you are certain that in a multithreaded application only one thread at a time ever executes any collections operation, then you can improve performance by disabling the collections thread lock. In other words, set **OS_COLL_THREAD_LOCKS** to **0**. You can do this when other threads are using ObjectStore entry points other than those involved with collections.

The **OS_THREAD_LOCKS** (see **OS_THREAD_LOCKS** on page 125) variable determines whether ObjectStore uses a lock to ensure that only one thread at a time can execute within ObjectStore code other than collections code.

If **OS_THREAD_LOCKS** is disabled, you cannot enable **OS_COLL_THREAD_LOCKS**. In other words, when **OS_THREAD_LOCKS** is set to **0**, ObjectStore cannot use thread locks in any portion of ObjectStore.

Caution

If you set **OS_COLL_THREAD_LOCKS** to **0**, and you cannot guarantee that only one thread at a time executes within collections code, you must enforce thread safety in some other way. If you do not enforce thread safety, and a multithreaded ObjectStore application has more than one thread executing within an ObjectStore collections entry point or the fault handler, then corruption of your nonpersistent data, ObjectStore's

nonpersistent data, or persistent data can occur. The corruption might or might not be detected when it occurs. Program failure can result.

An application cannot repeatedly enable and disable thread locks. Use the default, or establish the desired state before calling **objectstore::initialize**.

OS_COMMSEG_DIR

Default: **/tmp/ostore**

Pathname of the directory used for the communications segment. ObjectStore automatically places the commseg in the **/tmp/ostore** directory. If the UNIX file system containing **/tmp/ostore** is very small, it might be desirable to locate the communication segment 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** on page 98 for related information.

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 and OS/2

The operating system determines the location of the commseg in shared memory. You cannot change the location.

OS_COMMSEG_RESERVED_SIZE

Default: 8 MB

Specifies the maximum size, in bytes, to which the commseg (communications segment) can grow. If you set the maximum size to something less than the initial size, **OS_COMMSEG_SIZE**, ObjectStore automatically increases the reserved size to equal the initial size.

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

In previous releases, this variable was specified as **OS_COMMSEG_MAX_LENGTH**. This name is no longer accepted.

OS_COMMSEG_SIZE

Default: approx. 262,144 bytes, varies by platform and **OS_CACHE_SIZE** setting

Specifies, in bytes, the initial size of the communications segment (commseg). You must specify an integer that is a multiple of the page size.

The commseg is a preallocated region that holds ObjectStore internal data (global data, cache indexing info, and data that describes databases and segments used by the application).

The size of the commseg depends on the size of the cache file. Modify the size of the cache file as needed and ObjectStore adjusts the size of the commseg. The basic formula for commseg size requirements, roughly, is that you need

- At least 2000 bytes constant overhead
- Plus 64 bytes times the number of pages in the cache
- Plus 840 bytes for each segment you use or create
- Plus a bit more for each Server you use and for each database you use

For example, for 15,000 segments, assuming an 8 MB cache, which is 2048 four-kilobyte pages, you can expect to need about 12732840 bytes (12.7 MB) of commseg.

Specifying a value below the minimum size needed to store required structures in the communications segment has no effect.

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

An application can set its own commseg size with the API function **objectstore::set_commseg_size**.

OS_COMMSEG_START

Default: 0

Allows you to control the address of the commseg (communications segment). Normally, you do not need to set this variable; ObjectStore picks an address automatically.

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

OS_COMP_SCHEMA_CHANGE_ACTION

Default: error

Controls the severity of the error resulting from a type mismatch during library and compilation schema generation. You can set it to

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_DEBUG_C0000005

Default: false

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. To set this variable, specify any nonnull value except **0** to turn this variable on.

OS_DEBUG_LOCATOR_FILE

Default: 0

Instructs ObjectStore to send diagnostic information about the processing of the locator file to **stderr**.

When set to **0**, the default, no information is provided.

When set to **1**, ObjectStore generates a report every time the locator file is searched. This allows you to determine exactly what the input to the search is, which in turn lets you diagnose why the locator file is not providing the expected results.

If your application is a Windows GUI application, see **OS_STDOUT_FILE** on page 124.

OS_DEBUG_RECURSIVE_EXCEPTION

Default: false

On Windows only, when this is set, recursive exceptions are not handled by ObjectStore, so they can be debugged in a debugger.

OS_DEF_BREAK_ACTION

WIN

Default: false

Allows you to set a break point that obtains a stack trace before the stack is unwound. When you set this variable to **1**, ObjectStore reaches a hardcoded break point 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.

This environment variable cannot be used for systems running Windows NT on the Digital Alpha architecture.

OS_DEF_EXCEPT_ACTION

Default: not set

Controls what happens if an unhandled exception is signaled; this is useful in debugging unhandled exceptions. You can specify the following values:

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: • OS/2 — **DosExit (EXIT_PROCESS, 0x006600);** • Windows NT and 95— **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

WIN

Default: not set

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_DIRMAN_HOST

Default: not set

Specifies a rawfs host name that ObjectStore places at the beginning of every pathname that does not already begin with the *host*:: rawfs prefix. This variable provides a convenient way to toggle between rawfs and native file systems.

The only exception is for paths that start with **::**. In other words, when you set **OS_DIRMAN_HOST**, every path, except a path starting with **::**, is a rawfs database pathname. For example, if you specify **twinkie** for **OS_DIRMAN_HOST** and you specify **snoball:/myfile** for a pathname, ObjectStore treats it as though you had specified

twinkie::/myfile

However, if you specify **::/myfile**, **::blah** or **::../../foo** as the pathname, ObjectStore treats it as a file database on the local host. When **OS_DIRMAN_HOST** is set, this is the only way you can specify a file database.

If **OS_DIRMAN_HOST** is set, and **OS_DIRMAN_USE_SERVER_PREFIX** is set to **Yes**, ObjectStore builds the path using the value specified for **OS_DIRMAN_HOST**. If the result is **a::b:c**, ObjectStore interprets it as **b::c**. The **OS_DIRMAN_USE_SERVER_PREFIX** variable allows compatibility with previous ObjectStore releases. See **OS_DIRMAN_USE_SERVER_PREFIX** on page 106.

If you use only native file systems, do not set this variable; its use can cause difficulty in finding your databases or Server.

OS_DIRMAN_LINK_HOST

Default: not set

For use in resolving by-ID cross-database references, which have been obsolete since Release 1. Set the variable to the name of a Release 5 Server host to enable that Server as the clearinghouse for such references. The host must have a Release 5 rawfs with a directory called **/.ODI_compatibility_links**. In this directory there must be a link (possibly cross-server) to the actual database. Each link has a name of the form *x_y_z*, where *x*, *y*, *z* are the three words of the database ID, in hexadecimal, padded out to eight characters with leading zeros.

OS_DIRMAN_USE_SERVER_PREFIX

Default: false

Specifies an alternate interpretation of rawfs pathnames. Release 3 Directory Manager pathnames were allowed to have the form

a::b:/foo

In Release 4, ObjectStore interprets such names as

a::/foo

If this is not what you want, you can set the environment variable **OS_DIRMAN_USE_SERVER_PREFIX**. When this variable is set, ObjectStore interprets **a::b:/foo** as **b::/foo**.

When **OS_DIRMAN_HOST** is set and **OS_DIRMAN_USE_SERVER_PREFIX** is set, ObjectStore applies the setting of **OS_DIRMAN_HOST** first, and then applies **OS_DIRMAN_USE_SERVER_PREFIX**.

OS_DISABLE_PRE2_QUERY_SYNTAX_SUPPORT

Default: false

Intended for use with the ObjectStore C++ API. When set, causes the query translator to treat all uses of **[** and **]** found in query expression strings as array subscripting operations. Any setting of the environment variable is ignored if the application calls

os_coll_query::set_disable_pre2_query_syntax_support()

The use of **[** and **]** as the query element selection operator is discouraged. The use of **[%%** and **%%]** is preferred. In a future release of ObjectStore, support for **[** and **]** as the query element selection operator will be dropped. After you convert your sources to use the preferred form of the operator, set this variable to any nonnull value except **0**.

OS_DISPLAY_INSTALL_MISMATCHES

Default: false

Displays to **stdout** any mismatches found during 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** on page 124.

OS_ENABLE_PRE2_QUERY_SYNTAX_WARNINGS

Default: – (for **stderr**)

For use with the ObjectStore C++ API. When set to a nonnull string, enables warnings from the C++ API query translator about the use of obsolete query syntax for nested element selection [**(** and **)**]. The string should name the file to which the warnings should be written. Any setting of this variable is ignored if the application calls **os_coll_query::set_enable_pre2_query_syntax_warnings()**.

OS_ENABLE_REALTIME_COUNTERS

Default: false

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_EVICT_IN_ABORT

Default: false

Instructs ObjectStore to evict rather than relocate locked pages during an abort. This results in improved abort performance if relocation optimization is not in effect. The tradeoff is that pages must be fetched for subsequent operations. Specify any nonnull value except **0**.

OS_FORCE_DEFERRED_ASSIGNMENT

Default:false

When set to a nonzero value, causes all segments that are in enhanced format to use deferred assignment. If a segment uses deferred assignment, it cannot benefit from relocation optimization.

OS_FORCE_STANDARD_PRM_FORMAT

Default: false

When set to a nonzero value, causes all new databases to be created with the standard (old) PRM format. The standard format is necessary to ensure compatibility of cross-database pointers for applications that use databases created with ObjectStore prior to Release 5.0 and whose PRM format has not been upgraded with **osupgrm.**

See *ObjectStore Installation and License for Solaris* and the *ObjectStore C++ Interface Release Notes* for further information.

OS_FORCE_HANDLE_TRANS

OS_HANDLE_TRANS

UNIX

Default: false

Controls 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 **SIGSEGV**s, 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/2

OS/2 implements a subset of this feature. OS/2 ignores the value of **OS_FORCE_HANDLE_TRANS**. When you set **OS_HANDLE_TRANS**, 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 obtain a stack trace. OS/2 never runs the previous **SIGSEGV** handler if **OS_HANDLE_TRANS** is set. If you do not set up your own **SIGSEGV** handler before ObjectStore is initialized, the error message is SYS3175: Access Violation.

OS_IGNORE_LOCATOR_FILE

Default: false

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 but a small part of the locator file logic in the client is still executed. Consequently, if you have locator file debugging enabled, you still receive some diagnostic information. See **OS_DEBUG_LOCATOR_FILE** on page 103.

For information about the locator file, see Description of the Locator File on page 285.

OS_INBOUND_RELOPT_THRESH

Default: half the size of the persistent storage region

For databases that use enhanced-format PRM segments, sets the amount of process address space that can be used for immediate assignments. Values should be set in byte units.

If process address space already exceeds the threshold set, only immediate assignments that do not further increase process address space are allowed.

The default value is half the size of the persistent storage region. The PSR is controlled with **OS_AS_SIZE**, described in **OS_AS_SIZE** on page 93.

OS_INC_SCHEMA_INSTALLATION

Default: false

When set, 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.

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

An application performs incremental rather than batch schema installation if either of these conditions is true:

- The application has **OS_INC_SCHEMA_INSTALLATION** set.
- 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 *ObjectStore C++ API Reference*. For a discussion of the issues involved, see the *ObjectStore C++ API User Guide*.

OS_INHIBIT_TIX_HANDLE

Default: not set

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 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

Default: not set

OS_LANG_OVERRIDE 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 appropriate for the specific language locale. It is also 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, Object Design 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 Object Design 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 one of those 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**	Shift JIS Code (SJIS)	Shift JIS Code (SJIS)
Japanese **japanese.euc** **japan** **Japan** **jp_JP** **jp_JP.euc** **ja_JP** **ja_JP.euc**	Extended UNIX Code (EUC)	Extended UNIX Code (EUC)

OS_LIBDIR

Default: **OS_ROOTDIR\lib**

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 that is 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 default escape character for regular expressions in locator files is **$**, rather than the usual ****. 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 285.

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. See the *ObjectStore C++ API Reference*. See also Description of the Locator File on page 285.

OS_LOG_TIX_FORMAT

Default: false

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**

Specifies 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 **$OS_ROOTDIR/lib**, but under certain circumstances it might be elsewhere. This can happen when the machine where **$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_NB_LANA_NUM

Default: not set

Specifies the number of the default network protocol. Typically, you set this variable when your system is running more than one protocol. You do this to ensure that ObjectStore uses **NetBEUI** as its **NetBIOS** protocol.

On Windows, set **OS_NB_LANA_NUM** in the system environment so ObjectStore services have access to the proper value.

OS_NETWORK

Specifies the network dynamic link libraries (DLLs) to be loaded and used by all ObjectStore executables. On Windows, the default is

O4NETNSM,O4NETTCP,O4NETBIO

On OS/2, the default is

O4NETNP,O4NETTCP,O4NETSNA,O4NETBIO

You do not normally need to change the default setting. In other words, you do not usually set **OS_NETWORK**. Possible values for **OS_NETWORK** are

Possible values

- **O4NETTCP** — TCP/IP network DLL. Used for network communication.
- **O4NETBIO** — NetBIOS network DLL. Used for network communication.

Windows

On Windows, you can also specify

- **O4NETNSM** — Named shared memory DLL. Used for local interprocess communication.

OS/2

On OS/2, you can also specify

- **O4NETNP** — Named pipes DLL. Used for local interprocess communication.

In standard operation, ObjectStore automatically detects which networks are present, and **OS_NETWORK** does not need to be set. However, you can set **OS_NETWORK** to a comma-delimited list in situations where you want more control over the network used by ObjectStore.

Network list order

The order of the list is important. On Windows, when you set the **OS_NETWORK** variable, you must almost always specify **O4NETNSM** as the first network in the list. On OS/2, when you set the **OS_NETWORK** variable, you must almost always specify **O4NETNP** as the first network in the list. For local connections, the local network (**O4NETNSM** for Windows, **O4NETNP** for OS/2) 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 by hand before running the first ObjectStore application. Many other operations can fail if these DLLs are not specified at all or not specified first.

If you specify **O4NETBIO,** it must always be last. If the NetBIOS network is present, ObjectStore uses it in preference to other networks that might follow it in the list.

When to change the default

Suppose you originally installed ObjectStore on a system connected to a network. 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 **O4NETNSM** on Windows or to **O4NETNP** on OS/2 to prevent the TCP interface from loading and allow the Server to initialize normally.

On OS/2 systems, you might need to change the value of **OS_NETWORK** if you receive a message like the following when trying to run ObjectStore:

```
** Network error <err-0033-0705>SNA APPCerror: <maint-0033-0332>
AP_Comm>sub system_not_loaded **
```

ObjectStore signals an exception when it cannot enable a network protocol that it expects to enable, in this case, SNA networking. Set the **OS_NETWORK** environment variable to ignore the missing protocol or enable the SNA network.

If you are using TCP/IP on your OS/2 system, do so with a **set** statement in your **config.sys** file. For example:

set OS_NETWORK=O4NETNP,O4NETTCP

If you are using NetBIOS on your OS/2 system, the set statement is

set OS_NETWORK=O4NETNP,O4NETBIO

Windows

On Windows, if you have an unsupported TCP/IP stack on your machine, and you experience problems, set **OS_NETWORK** to **O4NETNSM,O4NETBIO** so that ObjectStore does not attempt to initialize or use the unsupported network.

OS_NO_MAPPED

Default: true

Instructs ObjectStore not to use mapped communication between the client and the Server. ObjectStore uses mapped communication only if the client and the Server are on the same host and the Server parameter **Allow Shared Communications** is set. To set this variable, specify any nonnull value except **0**.

OS_NOTIFICATION_QUEUE_SIZE

Default: not set

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, then 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 **16383.** If you set this variable to a value that is higher, ObjectStore uses **16383** as the value and not the value you set.

The API for setting the maximum length of the notification queue is **os_notification::maximum_notification_queue_length()**.

OS_OSDUMP_APPSCHEMA_PATH

Default:

Allows users to give a different path for **osdump.adb**.

OS_OSLOAD_APPSCHEMA_PATH

Default:

Allows users to give a different path for **osload.adb**.

OS_OSSG_CPP

Default: by platform

Sets the C preprocessor used by **ossg**. You can set this variable to a nondefault C preprocessor. The following table shows the default preprocessor on each platform.

OS/2	**icc**
UNIX	**cpp**
Windows	**cl**

OS_OUTBOUND_RELOPT_THRESH

Synonymous with **OS_RELOPT_THRESH**. See **OS_RELOPT_THRESH** on page 117 for a description of this environment variable.

OS_PORT_FILE

Default: by platform

The name of a ports file for network services. See Modifying Network Port Settings on page 51.

OS_PRINT_CLIENT_COUNTERS

Default: false

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

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 the Server parameter **Preferred Network Send Buffer Size**.

See **Preferred Network Send Buffer Size** on page 87 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 are rejected, which leads to reduced performance. Object Design recommends that you experiment by doubling the size until performance no longer improves.

OS_RELOPT_THRESH

C++

Defaults:
0x4000000
(64 MB)
0x10000000
(256 MB)
on AIX

The name is short for *relocation optimization threshold*. Use this variable to specify how much relocation information each segment in a database can store about every other segment in the database, in bytes. ObjectStore uses many criteria to determine whether or not to relocate a page. One factor is whether or not the total amount of pseudoaddress space assigned within the segment's relocation map has exceeded this threshold. If the threshold has been exceeded, ObjectStore suppresses the relocation.

Note that **OS_RELOPT_THRESH** is used to decide if outbound relocation is allowed. For information on controlling inbound relocation, see osprmgc: Trimming Persistent Relocation Maps on page 202.

The most common use of this environment variable is to disable outbound relocation optimization entirely by setting it to **0**. You can specify any number of bytes.

Relocation map

Every segment in an ObjectStore database contains a relocation map for the segment. If the database only consists of one segment, this is the relocation map for the entire database. However, if the database contains multiple segments, the relocation map for every segment in the database is stored in every other segment. This is true until the database grows to the size of **OS_RELOPT_THRESH**. When this occurs, relocation information for external segments is only added when a pointer to an object in another segment is added to the segment.

Performance

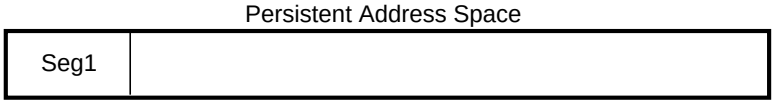
When the relocation maps get large and there are a large number of segments in the database, you lose the performance benefits of

having the segment relocation map repeated in each segment. In fact, the performance most likely deteriorates.

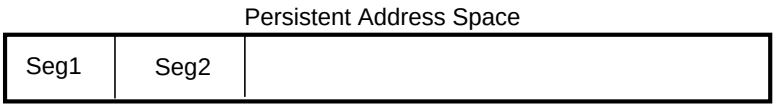
Setting **OS_RELOPT_THRESH** to **0** causes the ObjectStore client not to include any relocation information for other segments in the same database that are not being referenced by this segment. When the amount of relocation information becomes large, this saves the database reader the expense of reading in this additional information and also saves the database population program the expense of creating it.

Avoiding large relocation maps

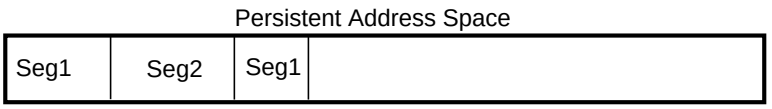
To avoid having large segment relocation maps, you can presize your segments with the **os_segment->set_size()** function. This limits the need for multiple entries in the relocation map for a segment. When a segment is created it reserves a default amount of persistent address space:



At this point there is only one entry in the relocation map for the segment. Roughly, this entry contains a location and a size. If you enlarge the segment through a persistent **new**, and the persistent address space above the end of the segment is available, only the size field of the relocation map entry needs to be updated. A new relocation map entry is not required. However, if the persistent address space above the segment at the time of the persistent **new** is not available and persistent address space looks like this:



Then after the **new**, persistent address space looks like this:



This forces the need for a new entry in the relocation map for Seg1. In addition, the entry is migrated to all the other segments in the database if the database has not reached the value of **OS_RELOPT_THRESH**.

Presizing the segment can help avoid this. You can use the ObjectStore utility **ossize** with the **-a** and **-c** options to see how many entries are in the relocation map for each segment of the database. This should help you determine if segment fragmentation is taking place.

Relocation optimization procedure

ObjectStore performs relocation optimization automatically whenever conditions permit. At a minimum, relocation optimization occurs when **objectstore**::**set_check_illegal_pointers** is not enabled.

What happens in relocation optimization is that ObjectStore tries to set up the transient address map (the in-use TRM) for a transaction so that all pseudoaddress values stored in a segment (that is, the process-neutral pointer values stored in the database) match up exactly with the process-specific virtual addresses that are used in the current transaction. In other words, inbound pointer relocation can be skipped for any page accessed in that segment because all pointers on any of the segment's pages already have correct address values.

Relocation optimization can result in a noticeable performance improvement, especially when many pages are retained in the client cache and reused in successive transactions.

For relocation optimization to be useful in as many cases as possible, it is important for the persistent relocation maps (PRMs) for each database segment to be as similar as possible. In other words, any two segments that both contain outbound pointers to the same segment should use the same pseudoaddress range to refer to that segment. This way, relocation optimization is effective regardless of the order in which segments are accessed in a transaction.

When relocation optimization is in effect, ObjectStore skips outbound pointer relocation when pages are evicted from the cache or when the current transaction is complete. Normally, it is during outbound pointer relocation that the ObjectStore client adds entries to a segment's PRM. This happens whenever an application updates objects in a segment to point to some other segment for the first time (or to a part of some segment that it never pointed to before).

When relocation optimization is in effect and a segment is modified in a committed transaction, ObjectStore takes the conservative approach of updating the segment's PRM to contain an entry for every entry in the in-use PRM. This includes any segments referenced in the current transaction, as well as any segments reachable from (that is, pointed to by) objects in those segments. The reason for this is that ObjectStore has no record of what changes were made to the segment, after a page in the segment became enabled for write access. It is possible that a pointer to a valid persistent object was copied from any other persistent object that was active in the transaction.

PRM bloat

The effect of this is referred to as *PRM bloat*. The PRM for a segment is likely to contain entries for segments that are never actually pointed to by that segment. Since address space is reserved for all entries in a PRM (when the segment is first accessed in a transaction), this condition might lead to excessive consumption of address space, especially as databases become large.

ObjectStore uses the environment variable **OS_RELOPT_THRESH** to put an upper bound on PRM bloat. The way this works is that when the sum of the sizes of all PRM entries in a segment exceeds the value of **OS_RELOPT_THRESH**, ObjectStore disables relocation optimization, and normal outbound pointer relocation is used to determine when the PRM needs to have new entries added to it.

If you know that a database your application uses is larger than the default value, but less than the maximum size of the persistent storage region, then you might choose to increase the value of **OS_RELOPT_THRESH** to keep the relocation optimization in force for larger databases.

On the other hand, if an application is trying to conserve address space, it might make more sense to set **OS_RELOPT_THRESH** to **0** to disable relocation optimization.

OS_RESERVE_AS

UNIX

Default: false

There is an optimization to ObjectStore that increases performance, sometimes by a very significant factor. ObjectStore uses this optimization when this environment variable is not set. This means performance is fast but there is potential for

confusion. ObjectStore does not use the optimization when this environment variable is set.

This optimization can cause trouble if your own program calls the **mmap** system call with **0** as the first argument, or if your program calls some 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 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: varies by platform

The top-level directory in the part of the file system hierarchy containing ObjectStore files 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 displayed:

```
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 the Server are on the same machine.

Defaults

The following table shows the defaults for **OS_ROOTDIR**.

Platform	***Default for* OS_ROOTDIR**
OS/2	The parent of the directory containing the **O4LOW.DLL** that is executing
Solaris 2	**/opt/ODI/OS5.0**
UNIX (except Solaris 2)	**/usr/local/ODI/OS5.0**
Windows	The parent of the directory containing the **O4LOW.DLL** that is executing

Windows

If ObjectStore was not installed in **C:\OSTORE**, you must set **OS_ROOTDIR** to the location of the directory where 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

OS_SCHEMA_KEY_HIGH specifies the high four bytes of a 64-bit schema key.

OS_SCHEMA_KEY_LOW specifies the low four 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 *ObjectStore C++ API Reference*. These environment variables 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**. 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.

These environment variables determine an application's schema key when an ObjectStore application attempts to access data in a schema-protected database, and either one 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.

Keep in mind that when the 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 like 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.
```

See also

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 *ObjectStore C++ API Reference*.

OS_SECURE_RPC_DOMAIN

Default: not set

Specifies a local domain name. For use 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_SNDBUF_SIZE

Default: 16384 bytes

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 Server parameter **Preferred Network Receive Buffer Size**.

See **Preferred Network Receive Buffer Size** on page 87 for additional information about the Server network buffer size. See further discussion about these parameters and how they affect performance in **OS_RCVBUF_SIZE** on page 116.

OS_STDOUT_FILE

WIN

Default: not set

Specifies the pathname 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 MessageBox.

To separate the output from several applications, you can set this variable to a pathname such as **C:\TEMP\DEBUG.%d**. ObjectStore substitutes the process ID in place of the **%d**.

OS_SUPPRESS_PRE2_QUERY_SYNTAX_WARNINGS

Default: false

Suppresses warnings when an application contains the pre-Release 2 element selection operator **[]**. To set this variable, specify any nonnull value except **0**.

OS_THREAD_LOCKS

Default: true

Determines whether ObjectStore uses a lock to ensure that only one thread at a time can execute within ObjectStore code other than collections code.

When this variable is **true** (set to a value other than **0**), ObjectStore uses the thread lock. This is the default.

If **OS_THREAD_LOCKS** is not set, you cannot enable **OS_COLL_THREAD_LOCKS** (see **OS_COLL_THREAD_LOCKS** on page 100). In other words, when **OS_THREAD_LOCKS** is set to **0**, ObjectStore cannot use thread locks in any portion of ObjectStore.

OS_TIX_BUFFER_SIZE

Default: 8192

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

Defaul:none:

Specifies the working directory for ObjectStore when an unhandled TIX exception causes ObjectStore to create a core dump. (Whether or not 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

Specifies a directory in which to place temporary files. If you do not set **OS_TMPDIR**, ObjectStore uses the path returned by the

Win32 API **GetTempPath()**. To set this variable, specify any nonnull value except **0**.

OS_TRACE_MISSING_VTBLS

Default: false

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
```

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 result in err_missing_vtbl errors later on

When you are debugging problems with missing vtbls, this helps determine which classes need vtbls. Any nonempty value except **0** enables the debugging message.

OS_TURN_ON_ENGLISH_MESSAGES

When set to **1**, forces the printing of English messages along with the catalog-retrieved language-specific message.

Chapter 4
Utilities

This chapter provides information about ObjectStore utilities. Many of these utilities are implemented using the **os_dbutil** class methods. See the *ObjectStore C++ API Reference*.

The following utilities are described in alphabetical order in this chapter:

Pathnames for utility executables

The pathname of the executable for an ObjectStore utility is

UNIX	**$OS_ROOTDIR/bin/***utility-name*
Windows and OS/2	**%OS_ROOTDIR%\bin***utility-name***.exe**

Earlier releases

In previous releases of ObjectStore, utilities were in the **/admin** and **/debugging** directories. These directories are obsolete.

FAT names on OS/2

On OS/2, if you install ObjectStore on a FAT file system, ObjectStore uses the FAT name for a utility if the usual name of the utility exceeds eight characters.

os_postlink: Fixing Vtbls and Discriminants

On cfront platforms, the **os_postlink** utility fixes vtbls and discriminants in your executable.

While **os_postlink** does not actually do anything on some platforms, Object Design recommends that you always call it from a makefile so that its absence does not cause a problem if you move the application to another platform.

Syntax

Use the **OS_POSTLINK** macro in your makefile to call the **os_postlink** utility.

$(OS_POSTLINK) *executable*

Description

When ObjectStore reads in an object with virtual functions, it supplies an appropriate vtbl pointer from the current application. This is called *vtbl relocation*.

When your application references a persistent object of a class with virtual functions, ObjectStore must fill in the vtbl pointer in the object. Virtual function tables are not stored in databases; they are part of your executable. To fill in the vtbl pointer, ObjectStore needs the address of the vtbl for the class.

During relocation, ObjectStore might need vtbls and discriminant functions. It finds them in tables that map class names to references to vtbls and discriminant functions. The schema generator generates a C++ source file (or object file for Visual C++) containing these tables that relate your schema to your application.

These tables are filled in during application link or postlink or at program start-up time, or some combination of these, depending on the platform. At each of these steps, the referenced vtbls and discriminants are searched for in the executable and, if found, are entered into the tables. At run time, ObjectStore can use these tables to find items for relocation.

On cfront platforms, the **os_postlink** executable performs this job. On some platforms, the compiler / linker does it. On some

platforms, this search might be done at run time based on the currently available DLLs.

API None.

osarchiv: Logging Transactions Between Backups

The **osarchiv** utility records all transaction activity for specified databases. You can run this utility interactively or in the background.

Syntax

osarchiv [*options***] -d** *directory* **[***pathname...***]**

-d *directory* Specifies the directory in which to create the archive log files. This is required.

pathname... Specifies a database or rawfs directory whose transactions you want to log. You can specify one or more pathnames. Pathnames can be on different Servers.

Databases can be file or rawfs databases.

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 pathname, you must specify the **-I** (I) option with an import file name.

Options

-a *archive-record-file* Specifies the pathname of the 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 is referred to as taking a snapshot. The archive record file is comparable to the incremental record file for **osbackup**.

-B *size* 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 megabyte. The default value is 1 MB.

-C

Enables the interactive command-loop feature. This feature is disabled by default.

-i *interval*

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 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 and 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 automatically taken. You can take a snapshot at any time that **osarchiv** is active by issuing the **x** command. See the command description for **x** on page 134.

-I *import-file*

Specifies the name of a file that contains a list of either file or rawfs database pathnames. 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 pathname per line. Leading and trailing white space is ignored.

If you specify the **-I** (uppercase I) option, you can also specify additional pathnames 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** on page 134. You cannot specify **-I-**.

-r

Instructs **osarchiv** to descend into 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 archiving file databases, specifying the **-r** option has no effect. You must explicitly specify each file database.

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 specified, it applies to the entire archive set.

-s *size*

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 megabyte.

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.

Commands

You can execute the following commands when you use **osarchiv** in interactive mode. The utility processes the command between snapshots.

a *pathname* — Adds the specified file database or rawfs database or directory to the archive set.

h — Displays on-line 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 automatically taken.

You can specify **i** without an integer to display the current interval.

You cannot take a snapshot of each transaction.

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 pathnames of the databases and rawfs directories in the archive set.

x — eXplicit — takes a snapshot as soon as you issue the command. This 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 you start **osarchiv**, the first snapshot records data modified by transactions that committed since the last time the **osbackup** or **osarchiv** utility was run.

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:

YYMMDDHH.*ext*

Variable	*Meaning*
YY	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.

Ensure 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

automatically begin to take snapshots of that database. 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.

OS/2 and Windows

On OS/2 and 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 that have 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 where 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 pathname of the file that **osarchiv** uses to record the segment 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 all rawfs databases found to the archive set. 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/96011216.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 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/96011216.aaa).
Writing backup volume #2 (/vancouver1/archives/96011216.aab)...

Display snapshot interval

> i
Snapshot interval is 5 seconds.

Change snapshot interval

> i 1m
> i
Snapshot interval is 60 seconds.

Remove member of archive set, 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 **osarchiv** utility	**> q** Closing volume #2 (/vancouver1/archives/96011216.aab). %
API	None.

osbackup: Backing Up Databases

The **osbackup** utility copies specified databases to another on-line location or to tape.

Syntax

osbackup [*options***] -f** *backup-image-file* **[-f** *backup-image-file***]...** *pathname* **...**

-f *backup-image-file*

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 1MB 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 NT systems, you specify a tape device with this syntax, **\\.\Tape0**, the standard Windows NT name for the first tape drive.

pathname...

Specifies a database or directory to be backed up. You can specify one or more pathnames. Pathnames can be on different Servers.

Options

-a

Aborts the backup operation if the utility cannot open the backup device. This 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.

Examples of failure to open the backup device are having a write-protected tape or no tape loaded.

-b *blocking-factor*

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*

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 megabyte. The default value is 1 MB.

-i *incremental-record-file*

Required. Specifies the incremental record file, a file that contains information about which databases have been backed up, and when they were backed up. The **osbackup** utility uses this information to determine which segments within a database have been modified since the last backup at a lower level. The utility then backs up only modified segments. 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*

Specifies the name of a file that contains a list of either file or rawfs database pathnames. 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 pathname per line. Leading and trailing white space is ignored.

If you specify the **-I** option, you can also specify additional pathnames on the command line.

-l *level*

Lowercase L 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 you did a level 2 backup on Monday, followed by a level 4 backup on Tuesday. A subsequent level 3 dump on Wednesday would contain all files modified or added since the level 2 (Monday) backup.

Backup is incremental at the segment level, meaning that a segment 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 segments in all specified databases.

-r

Instructs **osbackup** to descend into 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*

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, since end-of-media cannot be reliably detected 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*

Specifies the pathname 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**.

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 which segments 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.

Considerations

Incremental backup of file databases

To perform an incremental backup on a file database, you must have created the database using ObjectStore Release 4 or later. You cannot perform an incremental backup on file databases that were upgraded from releases of ObjectStore prior to Release 4.

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** (uppercase I) option.

Specifying an incremental record file

If you do not specify an incremental record file for your backup, **osbackup** creates one using a default pathname.

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 segment in the database. When you back up a database after compacting it, the **osbackup** utility copies each modified segment; this might be the entire database. Consequently, you might want to compact databases just before you perform a full backup.

Examples of Backing Up Databases

```
% osls -l vancouver::/foo.db
-rw-rw-r-- smith odi     231424 Dec 20 16:17 vancouver::/foo.db
%
```

Full backup of 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, it prompts 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 rawfs database to 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 databases listed in import file

% cat ./import_file
vancouver::/foo.db
/vancouver1/dbdir/foo.db
%

% osbackup -f ./img -i ./inc -l ./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 pathname

% osbackup -f ./img -i ./inc -l ./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 Thu Jan 12 15:50:10 1996

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 Thu Jan 12 15:50:21 1996

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 Thu Jan 12 15:50:41 1996

Do you wish to restore from any additional incremental backups?
(yes/no):
no
Closing volume #3 (./img2).
%

API None.

oschangedbref: Changing External Database References

The **oschangedbref** utility changes the external database references for the specified database.

Syntax

oschangedbref *db* **{***from* **| -n** *name1***} {***to* **| -n** *name2***}**

FAT name **oschange**

db	Specifies the database for which you want to change references.
from	Specifies the currently referenced database. This must be an absolute pathname that includes a Server host prefix.
-n *name1*	Specifies a relative pathname for the currently referenced database. You must use this option for names beginning with a hyphen.
to	Specifies the database to be referenced. This is a relative pathname or an absolute pathname, depending on how the original reference was defined.
-n *name2*	Specifies a relative pathname for the database to be referenced. You must use this option for names beginning with a hyphen.

Description

Before using this utility, run **ossize** to display the cross-database references in the database whose references you want to change. Carefully examine **ossize** output to determine how that database defines relative pathnames. Use this information to specify the *from* and *to* arguments.

Examples

UNIX

```
ossize loon:/dbs/db_0
[ ...]
External database pointers:
Relative name db_1, resolves to loon:/dbs/db_1

External references:
```

Relative name db_3, resolves to loon:/dbs/db_3

To change the reference to **db_3** to a reference to **db_7**, enter

oschangedbref loon:/dbs/db_0 loon:/dbs/db_3 loon:/dbs/db7

For the *to* argument, you can use the relative pathname instead of the absolute name. This is an equivalent command:

oschangedbref loon:/dbs/db_0 loon:/dbs/db_3 db7

Windows, OS/2

ossize me:h:\temp\t_1
[output omitted]

External database pointers:
Relative name t_3, resolves to me:h:\temp\t_3
Relative name t_0, resolves to me:h:\temp\t_0
Relative name t_2, resolves to me:h:\temp\t_2

External references:
Relative name t_3, resolves to me:h:\temp\t_3

To change pointers and references from **t_3** to **t_3.new**, enter

oschange me:h:\temp\t_1 me:h:\temp\t_3 me:h:\temp\t_3.new

or

oschange me:h:\temp\t_1 me:h:\temp\t_3 t_3.new

Example of moving to same directory

/a/b/db1 contains a reference to **/a/b/db2**.

If you move both **db1** and **db2** to a different directory, for example, **/e/f/g**, the reference is still valid because the result is

/e/f/g/db1
/e/f/g/db2

Both **db1** and **db2** are still in the same relative pathname.

If you move **db1** to **/e/f** and **db2** to **/e/f/g**, the result is

/e/f/db1
/e/f/g/db2

In this case, they are no longer in the same relative path. You need to use the **oschangedbref** utility.

Example of moving to different directories

/a/b/db1 contains a reference to **/a/b/c/db2**. In this case, **db1** refers to **c/db2**. So if you move **db1** to **/e/f** and move **db2** to **/e/f/c** the result is

/e/f/db1
/e/f/c/db2

The reference is still valid because **db1** still refers to **c/db2**.

Example of moving to different Server

/a/b/db1 on Server **green** contains a reference to **/a/b/db2** on Server **green**. You want to move **db1** to Server **red**. If you move them both to the same directory on Server **red**, the reference is still valid.

If you want to move only **db1** to Server **red** in directory **/x/y**, then you must use **oschangedbref** to change the reference from **db1** to specify the full pathname, including Server name, for **db2**.

When you store references, no Server name is attached until you use **oschangedbref** to specify it. The exception to this rule is if you use **os_database::set_relative_directory()**. See the *ObjectStore C++ API Reference*.

API

Class: **os_database**
Method: **change_database_reference**

oschgrp: Changing Database Group Names

The **oschgrp** utility changes the group name of the specified databases and directories.

Syntax

oschgrp [-R][-f] *group pathname* **...**

group	Specifies a group name or group number in a group ID file.
pathname **...**	Specifies the databases and/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	Indicates 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

This 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 pathname 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 pathname.

oschgrp can perform wildcard processing. See “Wildcards” on page 31.

UNIX

When operating on a rawfs database, you must enclose the wildcard in quotation marks ("") or precede it with a back slash (****) to keep the shell from interpreting wildcards.

oschgrp accepts a combination of rawfs pathnames and file pathnames.

/etc/group is the group ID file. You must be the owner of the database, or be the superuser.

API

Class: **os_dbutil**
Method: **chgrp**

oschhost: Changing Rawfs Link Hosts

The **oschhost** utility changes the host that a link in the rawfs points to.

Syntax

oschhost [-f][-R] *newhost pathname ...*

newhost Specifies the name of the new host for the specified rawfs link.

pathname **...** Specifies one or more rawfs links.

oschhost [*server_host***]** *old_link_host new_link_host*

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 host the link will point to.

Options

-f Forces execution. Errors are not reported.

-R Indicates that ObjectStore should change the host recursively for all specified directories.

Description

This utility operates only on rawfs links. The **oschhost** utility only changes 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 pathnames.

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

You must be the superuser to change the host for a rawfs.

API

Class: **os_dbutil**
Methods: **rehost_all_links** and **rehost_link**

oschmod: Changing Database Permissions

The **oschmod** utility changes the permission mode for the specified databases and directories.

Syntax

oschmod [-R][-f] *new_mode pathname ...*

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 pathnames.

Options

-R	Indicates that ObjectStore should change the permission recursively for all specified directories.
-f	Forces execution. Errors are not reported.

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 of the following modes (note that *execute* is meaningful only for directories):

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*] ...

who is a combination of

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

+	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

r	Read
w	Write
x	Execute

Omitting *permission* is useful only with **=**, to remove all permissions.

oschmod can perform wildcard processing. See “Wildcards” on page 31.

When you specify a file database, you cannot specify a remote file-server host in the pathname 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 pathname.

UNIX

When operating on a rawfs database, you must enclose the wildcard in quotation marks (" ") or precede it with a back slash (****) to keep the shell from interpreting wildcards.

API	Class: **os_dbutil** Method: **chmod**

oschown: Changing Database Owners

The **oschown** utility changes the ownership of specified databases and directories.

Syntax

oschown [-R][-f] *owner*[**.***group*] *pathname* **...**

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 pathnames.

Options

-R	Indicates that ObjectStore should change the owner recursively for all specified directories.
-f	Forces execution. Errors are not reported.

Description

This utility operates on rawfs databases and directories and file databases and directories.

When you specify a file database, you cannot specify a remote file-server host in the pathname 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 pathname.

oschown can perform wildcard processing. See “Wildcards” on page 31.

UNIX

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 operating on a rawfs database, you must enclose the wildcard with quotation marks ("") or precede it with a back slash (\) to keep the shell from interpreting wildcards. 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 pathnames and file pathnames.

API

Class: **os_dbutil**
Method: **chown**

oscmrf: Deleting Cache and Commseg Files

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***]**

hostname Specifies the host of the Cache Manager that you want to instruct to delete cache and commseg files. Defaults to the local host.

Description

It is always safe to run this 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 and OS/2

The Cache Manager does not use cache files or commseg files on OS/2 or Windows systems. However, you can use the **oscmrf** utility on these operating systems and specify hosts that do use cache files and commseg files.

Example

```
% oscmrf
Deleted 2 cache files and 2 commseg files.
%
```

API

Class: **os_dbutil**
Method: **cmgr_remove_file**

oscmshtd: Shutting Down the Cache Manager

The **oscmshtd** utility shuts down the Cache Manager on the specified host.

Syntax

oscmshtd [*hostname***] [***version***]**

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 **4**.

Description

Be sure to notify users before you shut down the Cache Manager.

Example

```
% oscmshtd
Shutting down Cache Manager process
%
```

API

Class: **os_dbutil**
Method: **cmgr_shutdown**

oscmstat: Displaying Cache Manager Status

The **oscmstat** utility displays status information about the Cache Manager process running on the specified host.

Syntax

oscmstat [*hostname***] [***version-number***]**

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 **4**.

Description

The information provided by the **oscmstat** utility is useful for debugging the storage system.

If you do not specify a host name, the default is the local host.

The **oscmstat** utility prints one line for 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
- A string that indicates what the thread is doing or what it most recently did
- 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 known to the Cache Manager. This is useful if you are trying to determine if files are in active use by ObjectStore, or are ObjectStore files no longer in use that can be deleted.

In **oscmstat** output, the second word of an ObjectStore file name is the name of the host that created and owns or owned the file. For example, for files named **objectstore_doolittle_commseg_8** and **objectstore_doolittle_cache_3**, the host name is **doolittle**.

The command **oscmstat doolittle** displays the files that the Cache Manager daemon on host **doolittle** currently knows about. If your file is *not* on the list, it is no longer in use, and can be removed with **oscmrf**.

If **oscmstat** reports that there is no Cache Manager running, it is safe to delete the file, as long as you are certain that **oscmstat** did not fail due to temporary network failure or something similar.

Example

UNIX

Output on a UNIX workstation typically looks like the following:

```
kellen% oscmstat
ObjectStore Release 5.0 Cache Manager, Version 9.0.[1]
Process ID 6444. Executable is path.exe.[2]
Host "kellen". Started at Sat May 20 14:54:05 1995
Soft Allocation Limit 0, Hard Allocation Limit 0. [3]
Allocated: free 80568320, used 5775360. [4]
Server host:      Client process ID:    Status for this host:
kellen 0 Initializing: constructor finished

There is 1 client currently running on this host:
Free files (cache):
    /tmp/ostore/objectstore_5_kellen_cache_1 (16777216) [5]
    /tmp/ostore/objectstore_5_kellen_cache_5 (8388608)
    /tmp/ostore/objectstore_5_kellen_cache_7 (8388608)
    /tmp/ostore/objectstore_5_kellen_cache_9 (8388608)
    /tmp/ostore/objectstore_5_kellen_cache_11 (8388608)
    /tmp/ostore/objectstore_5_kellen_cache_13 (8388608)
    /tmp/ostore/objectstore_5_kellen_cache_15 (8388608)
    /tmp/ostore/objectstore_5_kellen_cache_17 (8388608)
    /tmp/ostore/objectstore_5_kellen_cache_19 (8388608)
In-use files (cache):
    /tmp/ostore/objectstore_5_kellen_cache_3 (8388608)
Free files (commseg):
    /tmp/ostore/objectstore_5_kellen_commseg_18 (278528)
    /tmp/ostore/objectstore_5_kellen_commseg_16 (278528)
    /tmp/ostore/objectstore_5_kellen_commseg_14 (262144)
    /tmp/ostore/objectstore_5_kellen_commseg_12 (262144)
    /tmp/ostore/objectstore_5_kellen_commseg_10 (983040)
    /tmp/ostore/objectstore_5_kellen_commseg_8 (483328)
    /tmp/ostore/objectstore_5_kellen_commseg_6 (344064)
    /tmp/ostore/objectstore_5_kellen_commseg_4 (557056)
    /tmp/ostore/objectstore_5_kellen_commseg_2 (1622016)
In-use files (commseg):
    /tmp/ostore/objectstore_5_kellen_commseg_20 (262144)
Call Back Queue: Empty
Notifications [6]
```

Client PID	Queue Size	Received From Server	Received By Client	Pending	Overflows	Notifier State
13149	100	1814	1796	0	18	waiting_for_ notification
13145	43	904	876	0	5	waiting_for_ notification

kellen%

[1] Internal version number unrelated to ObjectStore release numbers.

[2] Operating system process ID of the Cache Manager process. The executable that you are running **oscmstat** from is identified by *path.exe*.

[3] The allocation limit parameters are as described in the parameter file.

[4] Total sizes of the used pool and the free pool.

[5] One line for each Server connection to the Cache Manager. This information is sometimes useful in debugging.

Cache file names end in odd numbers and commseg file names end in even numbers. The cache file whose name ends in **1** and the commseg file whose name ends in **2** go together. Likewise, the cache file whose name ends in **3** and the commseg file whose name ends in **4** go together, and so on.

One line for each client (ObjectStore application process) currently running on this host. For each client it gives the operating system process ID and user ID, the name of the client (assuming the client has called **objectstore::set_client_name()**), an internal version number that also has nothing to do with ObjectStore release numbers, and a virtual address within the Cache Manager that is useful in debugging the Cache Manager.

[6] For each client, the **oscmstat** output displays the following notification information:

- Process ID (PID) of the client.
- Size of the queue.
- **Received From Servers** 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.

- **Received By Client** is the number of notifications the client actually received. 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** is the number of notifications that are in the queue. This number can increase and decrease.
- **Overflows** is the number of notifications that were discarded because the queue was full. This number is for the lifetime of the client process and it only increases.
- **Notifier State** is for Object Design support purposes. It provides debugging information that is useful when you are familiar with the internal workings of the notification implementation.

OS/2 or Windows

On Windows and OS/2 systems, there are no cache or commseg files so there is no mention of them in **oscmstat** output.

API

Class: **os_dbutil**
Method: **cmgr_stat**

oscompact: Compacting Databases

The **oscompact** utility removes deleted space in specified databases or segments.

Syntax

oscompact {-dbs_to_compact *pathname* **... | -segments_to_compact** *pathname segment_number* **[***pathname segment_number***] ... }**
[-db_references *pathname ...***]**
[-segment_references *pathname segment_number*
[*pathname segment_number***] ...]**
[-compaction_threshold *percent_of_deleted_space***]**

Options

-dbs_to_compact *pathname* **...** Specifies one or more databases to compact. You must specify one of **-dbs_to_compact** or **-segments_to_compact**. You can specify both.

-segments_to_compact *pathname segment_number* Specifies one or more segments to compact. Identify each segment with its database pathname and segment number. You must specify one of **-dbs_to_compact** or **-segments_to_compact**. You can specify both.

-db_references *pathname* **...** Specifies one or more databases that contain pointers or ObjectStore references to the databases and segments being compacted.

-segment_references *pathname segment_number* Specifies one or more segments that contain pointers or ObjectStore references to the databases and segments being compacted. Identify each segment with its database pathname and segment number.

-compaction_threshold *percent_of_deleted_space* Specifies the minimum percent of deleted space that a segment must have to be compacted. Segments with less than the specified percent of deleted space are not compacted. When you do not specify this option, **oscompact** compacts any segment that has internal deleted space.

Description

The **oscompact** utility runs as an ObjectStore client process. After you compact a database,

- Access is usually faster so performance improves.
- You cannot decompact it.

You can obtain segment numbers by running the **ossize** utility or calling the API function **os_segment::get_number()**.

You can use the **oscompact** utility on both file databases and rawfs databases.

Compacting file databases

The segments 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 segment is required, the ObjectStore Server extends the host file to provide the additional space. The compactor permits holes contained in segments to be compacted for return to the allocation pool for the host file. This frees that space for use by other segments in the same database. However, since 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. See also **oscp** in oscp: Copying Databases on page 171.

Database size

Compacting a file database does not decrease its size, and might increase it to a small degree.

Compacting rawfs databases

The ObjectStore rawfs stores all databases in a single region, 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 segment in any database stored in the rawfs.

What the compactor does

The compactor compacts all C and C++ persistent data, including ObjectStore collections, indexes, and bound queries, and correctly relocates pointers and all forms of ObjectStore references to compacted data. ObjectStore **os_reference_local** references are relocated assuming they are relative to the database containing them. The compactor respects ObjectStore clusters, in that compaction ensures that objects allocated in a particular cluster remain in the cluster, although the cluster itself may move as a result of compaction.

Caution

When you have cross-database references, be sure to compact the databases together or use protected references. Not doing so can destroy references. If you run the **oscompact** utility on, for example, databaseA, then **os_reference**s from databaseB to databaseA are no longer valid. Alternatively, if you use protected references from databaseB to databaseA, then compacting databaseA does not cause a problem.

Backing up compacted databases

When you run the **oscompact** utility on a database, it has the potential to modify each segment in the database. When you back up a database after compacting it, the **osbackup** utility copies each modified segment; this might be the entire database. Consequently, you might want to compact databases just before you perform a full backup. However, as a safeguard against unexpected results, it is a good idea to back up databases just before you compact them.

Restrictions

You must observe the following data restrictions when using the compactor:

- Union discriminant functions require access to the representation to be compacted in order to run, and therefore cannot be compacted.
- Some data structures become invalid as a result of compaction. A classic example is a hash table that hashes on the offset of an object within a segment. Because compaction modifies these offsets, there is no way such an implicit dependence on the segment offset can be accounted for by compaction. Therefore, the compacted hash table becomes invalid. ObjectStore collections and indexes are valid after compaction.
- Since the ObjectStore **retain_persistent_addresses** facility requires that persistent object locations within a segment not vary, no client application using this facility and referencing segments to be compacted can run concurrently with the ObjectStore compactor.
- Transient ObjectStore references into a compacted segment become invalid after compaction finishes.
- The **oscompact** utility can run out of address space when compacting large databases. In these cases, you can compact your databases using the **-segments_to_compact** option.

Schema protection

When developing an application, if you are 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."
```

When deploying an application, if your end users need to use the **oscompact** utility on protected schema databases, you must wrap the utility in an application. This application must use the API to provide the key before using the **os_dbutil** class to call the utility. End users need not know anything about the key. For information about wrapping your application around an ObjectStore utility, see the class **os_dbutil** in the *ObjectStore C++ API Reference*.

API

Class: **objectstore**
Method: **compact**

oscopy: Copying Databases

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.

oscopy cannot be used to copy databases to file directories. It cannot copy segment-level permissions and it does not work with ObjectStore / Single. Instead, use the **oscp** utility.

Syntax

oscopy *source target*

source Specifies the ObjectStore file or rawfs database to be copied.

target Specifies the pathname for the copy. ObjectStore either creates this database or overwrites it. The target directory must be a rawfs file system.

oscopy -R *source_dir target_dir*

source_dir Specifies the pathname of the rawfs directory to be copied.

target_dir Specifies the target rawfs pathname. If this directory does not exist, ObjectStore creates it, if its base name exists. ObjectStore recursively copies the source directory into the target directory.

oscopy *source* **...** *target*

source Specifies the rawfs or file databases to be copied.

target Specifies the rawfs directory to contain the copies.

Options

-R Instructs **oscopy** to copy a directory recursively. You must specify a rawfs directory for both the source and destination pathnames. The top-level name of the destination pathname must exist before you issue **oscopy**.

Description

This command has three forms. The first copies a file or rawfs database to a rawfs file system. 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. You can specify either file or rawfs databases as sources. Copies must always be made to rawfs file systems.

Restrictions

You cannot specify wildcards in database pathnames.

Transaction consistency

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.

Database IDs

Native copy commands and the **oscopy** utility create copies with the same database ID as the original. This is important only if you have applications that rely on the uniqueness of these IDs. You can assign the copy a new, unique ID with **os_database::set_new_id()**.

Schema protected databases

When you copy a schema-protected database without specifying the schema key, the copy has the same **db_id** as the original. If you supply the correct schema key, the copy has a new **db_id**. In both cases, the copy has the same schema key as the original, and the key is frozen in the copy if it is frozen in the original.

Copying a rawfs database to a file database

Copying a rawfs database to a file database results in the loss of segment-level access control information.

Database size might change

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 just the right amount of space available for the copy of the database.

Variables that affect pathname interpretation

There are many conditions that can affect pathname interpretation:

- Settings of environment variables
- Whether or not there is a locator file
- Whether or not 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.

oscp: Copying Databases

The **oscp** utility makes a copy of an ObjectStore database.

Unlike **oscopy,** you can use **oscp** to copy ObjectStore / Single databases and to copy databases to file directories. Note, however, that **oscp** can produce an inconsistent database if other clients are updating the database while **oscp** is running. In these cases, use the **oscopy** command.

Syntax

oscp [-L *server_log*] *source target*

source Specifies the ObjectStore file or rawfs database to be copied.

target Specifies the pathname for the copy. ObjectStore either creates this database or overwrites an existing database of this name.

oscp -R [**-i**] *source_dir target_dir*

source_dir Specifies the pathname of the rawfs directory to be copied.

target_dir Specifies the target rawfs pathname for the copy of the rawfs directory. If this directory does not exist, ObjectStore creates it, if its base name exists. ObjectStore recursively copies the source directory into the target directory.

oscp [**-Ri**] *source* **...** *target*

source Specifies the rawfs databases to be copied.

target Specifies the rawfs directory to contain the copies.

Options

-i Instructs **oscp** to prompt you to confirm whether or not to overwrite databases or directories at existing pathnames. If *target* does not exist, you do not see this prompt.

-L *server_log* When specified, the named file is used for the Server log file. When unspecified, a temporary file is used.

This option is only applicable when you are running the utility as an ObjectStore / Single application. If the file already exists, it must be a properly formed Server log.

-R Instructs **oscp** to copy a directory recursively. You must specify a rawfs directory for both the source and destination pathnames. The top-level name of the destination pathname must exist before you issue **oscp**.

Description

Restrictions

- You cannot specify wildcards in database pathnames.
- Because **oscp** uses segments as a unit in copying, it is possible for **oscp** to produce an inconsistent database if other clients perform updates while **oscp** is running.

Using native copy commands

The **oscp** utility contacts the Server to ensure that the database being copied is transaction-consistent and fully up-to-date. The native copy commands do not do this. Therefore, you should only use native copy commands if the Server for the file database you want to copy has been shut down. (You cannot, of course, use native copy commands to copy a database to or from a rawfs.)

Using native copy commands sometimes produces a database

- In an inconsistent state if the database was copied during propagation of data from the log
- In a consistent but out-of-date state if the effects of some transactions have not yet been propagated from the log to the database

Attempting to operate on an inconsistent copy fails, signaling err_inconsistent_db.

Database IDs

Native copy commands and the **oscp** utility create copies with the same database ID as the original. This is important only if you have applications that rely on the uniqueness of these IDs. You can assign the copy a new, unique ID with **os_database::set_new_id()**.

Schema-protected databases

When you copy a schema-protected database without specifying the schema key, the copy has the same **db_id** as the original. If you supply the correct schema key, the copy has a new **db_id**. In both cases, the copy has the same schema key as the original, and the key is frozen in the copy if it is frozen in the original.

Copying a rawfs database to a file database

Copying a rawfs database to a file database can result in the loss of segment-level access control information. This can happen because file databases do not maintain this information. The **oscp** utility issues a warning after copying a database if the source database had segment-level protections that could not be copied. If the source database is not using segment-level access control, nothing is lost and a warning is not displayed.

Database size might change

Your database might appear to have a different size after you use **oscp** 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 **oscp**, the size of the database is set. The Server can make just the right amount of space available for the copy of the database.

Variables that affect pathname interpretation

There are many conditions that can affect pathname interpretation.

- Settings of environment variables
- Whether or not there is a locator file
- Whether or not 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.

Examples

These examples take two ObjectStore environment variables into account.

- **OS_DIRMAN_HOST** — Specifies a rawfs host name that ObjectStore places at the beginning of every pathname that does not already begin with the *host***::** rawfs prefix.
- **OS_DIRMAN_USE_SERVER_PREFIX** — When this variable is set, ObjectStore interprets **a::b:/foo** as **b::/foo**. When it is not set, ObjectStore interprets **a::b:/foo** as **a::/foo**.

When **OS_DIRMAN_HOST** is set and **OS_DIRMAN_USE_SERVER_PREFIX** is set to **Yes**, ObjectStore applies the setting of **OS_DIRMAN_HOST** first, and then applies **OS_DIRMAN_USE_SERVER_PREFIX**.

Simple copy

Suppose neither variable is set and you invoke **oscp** as follows:

```
oscp /source/db1 /target/db2
```

If you do this on a host named **atlas**, then the full pathname of the copy is **atlas:/target/db2**, which is a Server-relative pathname. A Server-relative pathname is the operating system pathname as opposed to an ObjectStore rawfs pathname.

OS_DIRMAN_HOST

Now suppose that you set **OS_DIRMAN_HOST** to **mars**. If you execute the two commands below on **atlas**, each command produces the same result.

```
oscp /source/db1 /target/db2
```

```
oscp atlas:/source/db1 /target/db2
```

The first command line is the same as the simple copy example above, but the result is different from the previous example. ObjectStore interprets **/source/db1** as **mars::/source/db1**. The full pathname of the copy is **mars::/source/db1**.

In the second command, ObjectStore interprets **atlas:/source/db1** as **mars::atlas:/source/db1** and then as **mars::/source/db1**. This is the default interpretation when **OS_DIRMAN_USE_SERVER_PREFIX** is not set.

OS_DIRMAN_USE_SERVER_PREFIX

In the next example, **OS_DIRMAN_HOST** is not set, but **OS_DIRMAN_USE_SERVER_PREFIX** is set to **Yes**. Invoke the following command on **atlas**:

```
oscp mars::atlas:/source/db1 /target/db2
```

ObjectStore interprets **mars::atlas:/source/db1** as **atlas:/source/db1**. The full pathname of the copy is **atlas:/target/db2**.

Both variables set

In the last example, **OS_DIRMAN_HOST** is set to **mars** and **OS_DIRMAN_USE_SERVER_PREFIX** is set to **Yes**. Invoke the following **oscp** command on **atlas**:

```
oscp atlas:/source/db1 /target/db2
```

ObjectStore interprets **atlas:/source/db1** in two steps.

1 ObjectStore applies the setting of **OS_DIRMAN_HOST**, so the result is **mars::atlas:/source/db1**.

2 ObjectStore applies the setting of **OS_DIRMAN_USE_SERVER_PREFIX**, so it interprets **mars::atlas:/source/db1** as **atlas:/source/db1**.

The full pathname of the copy is **mars:/target/db2**.

API

Class: **os_dbutil**
Method: **copy_database**

osdf: Displaying Rawfs Disk Space Information

The **osdf** utility shows the amount of used and available disk space for the rawfs on the specified Server.

Syntax

osdf *hostname*

hostname	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 how much growth room is available.

Example

osdf elvis

Filesystem	kbytes	use	avail	capacity
elvis	95749	533	95215	0%

API

Class: **os_dbutil**
Method: **disk_free**

osdump: Dumping Databases

The **osdump** utility dumps to ASCII a database or group of databases, and generates the source for a loader capable of creating equivalent databases.

Syntax

osdump [**-pseudo**] [**-emit**] *pathname ...*

pathname ... One or more pathnames, separated by spaces, specifying the database or databases to be dumped, or (if **-emit** is supplied) specifying the database or databases for which loader source is to be emitted.

The file name at the end of each path must have the form *filename***.db**; that is, it must have the extension **.db**.

Options

-emit Tells **osdump** to generate the source code for a loader executable for the specified databases. To generate an ASCII dump of the specified databases, do *not* specify **emit**.

-pseudo When used with **-emit**, tells **osdump** to generate the files **ldrcls00.h** and **ldrcls00.cpp**, which contain pseudo declarations of the classes in the dumped databases. When used without **-emit**, this switch has no effect.

When you build the loader executable, you can specify the schema of the databases with either of the following:

- Original C++ code used to create the databases
- **ldrcls00.h** and **ldrcls00.cpp**

If you want to use the original C++ code, do *not* supply **-pseudo**, and change **#include "ldrcls00.h"** in generated code to include the original **.h** files instead.

If you want to use **ldrcls00.h** and **ldrcls00.cpp**, supply **-pseudo**, and do *not* change the include lines.

Description

Each execution of **osdump** does one of the following:

- Dumps to an ASCII file the contents of a database or group of databases.
- Generates the source files for a loader executable capable of creating, given the ASCII as input, an equivalent database or group of databases. **osdump** also generates a makefile for your platform that allows you to build the loader executable with a single **make** command.

For each database or group of databases you want to dump, execute **osdump** twice:

- Once without **-emit**, to generate the ASCII for the databases
- Once with **-emit**, to generate the source for a dumper tailored to the databases

The dumped ASCII has a compact, human-readable format. It is editable with tools such as perl, awk, and sed. You can use edited or unedited ASCII as input to the loader.

The schema for the dumped databases is stored in a remote schema database associated with the dump.

See also osload: Loading Databases on page 196.

Default compared to customized dump and load

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. To determine when to customize, see When Is Customization Required? on page 181. To learn how to customize, see Chapter 8, Dump/Load Facility, in the *ObjectStore Advanced C++ API User Guide*.

Databases with unions, pointers-to-members, or multidimensional arrays cannot be dumped by this utility.

Generated ASCII files

When invoked without **-emit**, **osdump** generates the following files in the current directory:

- *filename*.**dmp** for each dumped database, *filename*.**db**
- **db_table.dmp**, which records information about the dumped databases

See Default Dumper ASCII Format on page 182 for a description of the layout of the generated ASCII file.

Generated source and makefiles

If **osdump** is invoked with **-emit**, it generates the following files in the current directory:

- **ldrdef00.h**: **struct**s to hold loaded data before class construction.
- **ldrcls00.h**: Generated only if **-pseudo** is supplied together with **-emit**. Default declarations for the classes in the dumped databases. These declarations contain class constructors that take the corresponding **struct**s from **ldrdef00.h** as an argument.
- **ldrldr00.h**: Declarations of the classes needed for loading each of the classes in the dumped databases.
- **ldrcls00.cpp**: Generated only if **-pseudo** is supplied together with **-emit**. Default implementations of the constructors declared in **ldrcls00.h**.
- **ldrldr00.cpp**: Implementation of the classes needed for loading each of the classes in the databases.
- **ldrsch00.cpp**: **OS_MARK_SCHEMA_TYPE()** calls for each class in the databases.
- **ldrmai00.cpp**: The **main()** file for **osload**.
- **makefile.unx**: Build file for UNIX.

- **makefile.w32**: Build file for Windows.

If files with these names exist in the working directory, **osdump** overwrites them.

To build the loader from these files, use one of the generated makefiles. On UNIX, use the **make** utility and the **osdump**-generated makefile **makefile.unx**. On Windows, use **nmake** and the **osdump**-generated makefile **makefile.w32**.

Default Equivalence

This section defines *equivalence* for databases dumped and loaded by the default dump and load processes.

Roughly speaking, two databases are equivalent if every object in one database has a corresponding, equivalent object in the other database, where two objects are equivalent if

- The fundamental values they contain, directly or indirectly, are the same.
- The pointers or references they contain refer to corresponding objects (that is, if an object in a dumped database points to o, the corresponding object in the loaded database points to the object that corresponds to o).

More precisely, two databases, db1 and db2, are equivalent if and only if there is a one-to-one mapping, *map*(), between objects in db1 and objects in db2 such that for every object, o1, in db1, o1 is equivalent to *map*(o1).

Two objects, o1 and o2, are equivalent (according to *map*()) if and only if all the following hold:

- o1 and o2 have the same type.
- If the type of o1 is fundamental, o1 and o2 have the same value.
- If o1 is a pointer or (C++) reference, *map*(the referent of o1) is the referent of o2.
- If o1 is an array, o1 and o2 have the same cardinality, and for every *index-list*, o1[*index-list*] is equivalent to o2[*index-list*].
- If the type of o1 is a non-ObjectStore class, then for every member, *m*, of o1, o1.*m* is equivalent to o2.*m*.
- If o1 is an ObjectStore reference, *map*(the referent of o1) is the referent of o2.

- If o1 is an ObjectStore collection, o1 and o2 have the same cardinality, and for each element, e, of o1, e has the same count in o1 as *map*(e) does in o2.
- If o1 is an ordered ObjectStore collection, then for each element, e, of o1, e has the same position in o1 as *map*(e) does in o2.
- If o1 is an ObjectStore index, o1 and o2 have the same path and index options, and *map*(the collection that o1 indexes) is the collection that o2 indexes.
- If o1 is an ObjectStore cursor, o1 and o2 have the same cursor options, and *map*(the element at which o1 is positioned) is the element at which o2 is positioned.
- If o1 is an ObjectStore database root, o1 and o2 have the same root name, and *map*(o1's associated entry-point object) is o2's associated entry-point object.
- If o1 is an ObjectStore index path, o1 and o2 represent the same paths.

When Is Customization Required?

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 segment. 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, for example, **objectstore::get_pointer_numbers()** might return different values for them.

The default dumper and loader take into account the first implication, and the loader automatically adjusts all pointers in loaded databases to use the new locations.

The default dumper and loader also take into account the second implication for ObjectStore collections with hash-table representations. Since a dumped collection element hashes to a different value than the corresponding loaded element does, their

hash-table slots are different. So the facility 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) should also be dealt with through customization.

See Chapter 8, Dump/Load Facility, in the *ObjectStore Advanced C++ API User Guide*.

Performance

To enhance efficiency during a dump, database traversal is performed in address order whenever possible. To enhance efficiency during loads, loaders are generated by the dumper and tailored to the schema involved. This allows the elimination of most run-time schema lookups during the load.

Default Dumper ASCII Format

Each **db_table.dmp** file has the format for **database_table** described below.

For each dumped database, *filename*.**db**, *filename*.**dmp** has the format shown for **database**, below.

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 roots segments*

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 [*cluster_size***] {** *objects* **}**

```
objects ::=
    object*

object ::=
    id ( type ) value

id ::=
    <database_index,segment_number,offset>

offset ::=
    integral value denoting the byte offset of an object
        within its segment

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

integral ::= any non-floating-point decimal number

floating_point ::=
    any floating-point decimal number

pointer_value ::=
    any hex unsigned integral number
```

string **::=**
"s" where s is any sequence of printable ascii characters
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, 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 | *collection_with_representation_policy* |
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

```
collection_with_representation_policy::=
    [ behavior cardinality collection_type representation_enum ]
        { [representaion_enum]* }

dummy_cursor::=
    [ D ]

cursor::=
    [ C collection_reference safe_flag ]

collection_reference::=
    reference

safe_flag::=
    integral

cursor_with_index::=
    [  C collection_reference safe_flag
        { 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: Displaying Class Names in a Schema

The **osexschm** utility lists the names of all classes in the schema referenced by the specified database.

Syntax

osexschm [-detail] *pathname*

-detail	Describes the structure of every class in detail.
pathname	Specifies a file or rawfs database.

Description

For each class, **osexschm** indicates whether an object of the class type can be persistently allocated.

Schema protection

When developing an application, if you are 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."

API

None.

osgc: Garbage Collection Utility

Garbage collection frees storage associated with persistent objects that are unreachable. Applications can continue to use a database while garbage collection is in process.

The command line utility for collecting garbage is **osgc**. Invoke this tool with the following format:

osgc [*options* **]** *database_name*

You can specify the following options:

Option	***Description***
-seg *segment_id*	Collects garbage from only the specified segment. 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.
-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.
-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.
-transactionPriority *n*	Specifies the transaction priority associated with transactions started by the tool. The Server uses this specification when it must determine which transaction 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.

-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. 1 lists the number of objects per segment that would be destroyed. 2 is not currently supported. 3 lists the location of each GC candidate. 4 lists the roots of garbage graphs. Level 4 can require intensive computations.
-statistics	Displays statistics for the garbage collection operation. This includes the total number of reachable objects and the total number of garbage objects.

Performing Garbage Collection in a Database

The ObjectStore persistent garbage collector (GC) collects unreferenced objects and ObjectStore collections in an ObjectStore database.

Persistent garbage collection frees storage associated with objects that are unreachable. It does not move remaining objects to coalesce the free space. (See oscompact: Compacting Databases on page 164)

The GC performs its job in two major phases. In the mark phase, the GC identifies the unreachable objects. In the sweep phase, the GC 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 to be garbage collected.

C++ Usage note

Normally, databases resulting from ObjectStore applications written in C++ will not require garbage collection since all storage allocation is handled explicitly.

osgc can be useful as a debugging tool. For example, if unreferenced objects are being harvested, it's an indication of a persistent memory leak. The identity of these objects can be a clue to the root of the problem.

Restriction

Do not use **osgc** with applications that rely on cross-database pointers, The garbage collector operates on one database at a time. References to one database from another are not detected and objects pointed to by references from other databases are seen as unreferenced and therefore removed.

Applications can continue to use a database while persistent GC is in progress. GC locks portions of a segment as needed, just as if it were just another application. In this way, the GC minimizes the number of pages that are locked and the duration for which the locks are held. Also, the GC retries operations when it detects lock conflicts.

By default, the GC runs with a transaction priority of zero. Consequently, it is the preferred victim when the Server must terminate a transaction to resolve a deadlock. At a later time, the GC 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 GC cannot acquire a lock because of a timeout, it retries the operation at a later time.

osglob: Expanding File Names

The **osglob** utility performs ObjectStore file name expansion.

Syntax

osglob *wordlist*

wordlist Specifies strings, such as rawfs pathnames, containing wildcards that you want to expand into all matching pathnames.

Description

The **osglob** utility can perform wildcard processing similar to regular expression wildcards *****, **?**, **{}**, and **[]**.

UNIX When operating on a rawfs database, you must enclose the wildcard in quotation marks ("") or precede it with a back slash (****) to keep the shell from interpreting wildcards.

API Class: **os_dbutil**
Method: **expand_global**

oshostof: Displaying Database Host Name

The **oshostof** utility displays the host of the specified database to standard output.

Syntax

oshostof *pathname*

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 pathname syntax is supported, including the **OS_DIRMAN_HOST** compatibility feature.

When you specify a pathname that is a symbolic link **oshostof** displays the host of the database that the link points to.

When you specify the pathname of a Server-remote database the **oshostof** utility returns the name of the host where the database resides.

Examples

A typical use is as follows:

ossvrchkpt 'oshostof a/b/c'

API None.

osln: Creating Links in the Rawfs

The **osln** utility creates a symbolic link in the rawfs hierarchy.

Syntax

osln *pathname linkname*

pathname	The pathname of the rawfs directory or database that you want to point to.
linkname	The pathname of the rawfs directory or database that is the new link. It points to *pathname*.

Description

Different links can point to the same rawfs pathname.

To indicate hosts, specify pathnames 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 or not the Server was previously connected to in the link chain. When the client reaches the sixteenth link, ObjectStore displays 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 displays 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 one. 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. For example, A is a link to B, and B is either directly or indirectly a link to A.

When 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 do not need to 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 that 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: Removing Databases and Rawfs Links on page 222.

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
```

API

Class: **os_dbutil**
Method: **make_link**

osload: Loading Databases

To load a database or group of databases from an **osdump**-generated ASCII file, build the executable **osload** from the corresponding **osdump**-generated source.

On UNIX, use the **make** utility and the **osdump**-generated makefile **makefile.unx**. On Windows, use **nmake** and the **osdump**-generated makefile **makefile.w32**.

The utility **osload** creates a database or group of databases given **osdump**-generated ASCII as input. The resulting databases are equivalent to the ones from which the ASCII was produced.

Syntax

osload [**-cwd**] **db_table.dmp** *pathname* ...

db_table.dmp	Database table dump file generated by **osdump**. Records information about the dumped databases.
pathname **...**	One or more pathnames, separated by spaces, specifying the ASCII dump files to be loaded.

Options

-cwd	Tells **osload** to recreate databases in the current working directory.

Description

For given ASCII input, the databases created by **osload** have the same file names as the databases from which the ASCII was generated (as stored in **db_table.dmp**).

If switch **-cwd** is not set, then the databases have the same pathnames (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), **osload** aborts.

-cwd forces **osload** to ignore paths from **db_table.dmp** and create the databases in the current working directory.

osls: Displaying Directory Content

The **osls** utility lists the contents of the specified directory.

Syntax

osls [-dlRsu] *pathname ...*

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 pathname includes links, ObjectStore identifies the pathname as the pathname 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 pathnames. It accepts a combination of rawfs pathnames and file pathnames.

When you specify a local directory, you cannot specify a remote file-Server host in the pathname 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 pathname.

This utility can perform wildcard processing using regular expression wildcards *****, **?**, **{}**, and **[]**.

UNIX

When operating on a rawfs database, you must enclose the wildcard in quotation marks ("") or precede it with a back slash (****) to keep the shell from interpreting wildcards.

API

Class: **os_dbutil**
Method: **list_directory**

osmkdir: Creating a Rawfs Directory

The **osmkdir** utility creates a directory in the rawfs.

Syntax

osmkdir [-p] [-m *octal-mode***]** *directory*

Options

-p	Indicates that ObjectStore should create any missing directories that are needed to make the specified directory path exist.
-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.
directory	Specifies a rawfs directory pathname.

Description

You can also use **osmkdir** to create a nonrawfs directory. When you create a nonrawfs directory, you cannot specify a remote file-server host in the pathname 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 pathname. If you specify the **-p** option, it works if the native utility supports that feature.

API

Class: **os_dbutil**
Method: **mkdir**

osmv: Moving Directories and Databases

The **osmv** utility moves a database, directory, or link.

Syntax

Rawfs

osmv [-fi] *p1 p2*
osmv [-fi] *p1*... *pn dir*

File databases

osmv [-fi] *p1 p2*

p1 *p1 ... pn*	Specifies the pathname of a file database or a rawfs database, link, or directory that you want to move.
p2	Specifies the new pathname for the file database or rawfs database, link, or directory. If *p2* is a link to a directory, ObjectStore places *p1* 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 *p1* and *p2* are rawfs databases or links, **osmv** moves (changes the name of) *p1* to *p2*. If *p2* already exists, the utility removes it and then moves *p1* to *p2*. If *p1* is a rawfs directory, then *p2* must not already exist. **osmv** moves (changes the name of) the *p1* directory to the *p2* 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 *p1* to file database *p2*.

Procedure

When moving rawfs databases to another Server, the **osmv** utility moves an item by doing the following:

1 Remove the destination, if it exists.

2 Copy the source to the destination.

3 Remove 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.

Fix external pointers and references

After you move a database, you need to use the **oschangedbref** utility to fix external pointers and references. See oschangedbref: Changing External Database References on page 146.

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, then 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 pathname of the file database.

osmv can perform wildcard processing using regular expression wildcards *****, **?**, **{}**, and **[]**.

UNIX

When operating on a rawfs database, you must enclose the wildcard in quotation marks ("") or precede it with a back slash (\) to keep the shell from interpreting wildcards.

API

Class: **os_dbutil**
Method: **rename**

osprmgc: Trimming Persistent Relocation Maps

The osprmgc utility prevents PRM bloat.

Syntax

osprmgc [-q] [-r] [-n *N***] [-t** *keyword***]** *database-name*

Options

-q Quiet mode does not print results after every segment, but provides a report of total ranges found and total pages collected.

-r Read-only mode calculates how many ranges can be collected, but does not do the collection. It runs MVCC.

Another circumstance that produces a read-only report is if you run the utility on a database for which you have only read permission. In such a case, however, the utility does not run MVCC unless **-r** was specified.

-n *N* Specifies that **osprmgc** examine segment *N* only. This allows you to take advantage of the PRM reduction in one segment without subjecting the entire database to garbage collection.

-t *keyword* This option accepts the following keywords as values.

remove_whole_ranges – Default. Removes whole unused PRMEs. This is the only possible setting for use with immediate address-space assignment.

shrink_ranges – Removes whole unused PRMEs and shrinks any remaining non-huge PRMEs.

This setting minimizes address space as much as possible without increasing the number of PRMEs.

split_ranges – Removes whole unused PRMEs. Any remaining PRMEs are split and the unused space is removed. This setting best minimizes address-space usage, but can increase the number of PRMEs in existence.

coalesce_ranges – This setting can increase address-space usage. It is the best setting to reduce the number of PRMEs.

Description

The **osprmgc** utility reduces the size of persistent relocation maps (PRMs) by removing unnecessary persistent relocation map entries (PRMEs). The PRM governs the translation of pseudoaddresses, for persistent pointers, to process addresses. While the PRMs are stored in a compact form, the maps useful to an application, transient relocation maps (TRMs), are larger and consume transient heap memory. Also, large TRMs have a significant impact on persistent address space in the process.

The PRM grows whenever you add a pointer that is not yet translated by an existing PRME. Outbound relocation adds the necessary entry. The issue is that although pointers come and go (and once they are gone, the corresponding PRME might no longer be needed), the PRMEs are not normally removed. The PRM does not normally shrink.

If you are using relocation optimization, unnecessary PRM expansion can occur, since the entire lot of translations is taken in the interest of performance. That is, when you use relocation optimization for the sake of better performance, the PRM size can increase dramatically.

The **osprmgc** command-line utility is a PRME garbage collector that shrinks the size of the PRM so that it translates only the pointers currently existing in the segment. To decide when an entry should be removed, the utility looks at every data page in the segment to ensure that the entry is no longer needed.

The benefits of running the **osprmgc** utility are that it

- Reduces the amount of memory consumed by the transient relocation map used by the application
- Increases the speed of transfer from PRM to TRM and back
- Reduces the consumption of persistent address space

The utility uses one transaction per segment. It reports by segment and also provides a total number of ranges found and collected per database.

Additionally, an embedded form of the utility exists in an **os_dbutil** version. This is particularly useful for databases with discriminant unions. The format is as follows:

```
struct os_prmgc_options {
    os_boolean flag_quiet; // -q, default is false
    os_boolean flag_read_only; // -r, default is false
    os_boolean flag_one_segment; // -n, default is false
    os_unsigned_int32 one_segment_number; // the N in -n N
    os_prmgc_type prmgc_type; //-t default is remove_whole_ranges
};
```

Discriminant union considerations

If you have a database with discriminant unions, you must perform PRM garbage collection using the **os_dbutil** form, and link in the necessary discriminant functions.

Both the command-line and embedded versions of the utility use a streaming fetch policy. The embedded version ensures that the policy is restored to its original state (if different) to minimize impact on the application.

Environment Variables

By default, when a segment is put in use, the address space assignment for that segment is immediate when inbound relocation optimization is possible. The default becomes deferred assignment either because inbound relocation optimization is not possible, or because trying to get immediate assignment would increase the amount of assigned address space above half of **OS_AS_SIZE**.

The defaults are effective for the large majority of conditions, but for extreme cases, there are override mechanisms. The default behavior can be overridden by the following environment variables:

- **OS_FORCE_STANDARD_PRM_FORMAT**
- **OS_IMMEDIATE_THRESH**
- **OS_MAX_IMMEDIATE_RANGES**

See *ObjectStore Management* for a description of the use of these environment variables. Note that the existing environment variable **OS_RELOPT_THRESH** does not affect this choice. It is only used to decide if outbound relocation optimization is allowed.

API

Class: **os_dbutil**
Method: **osprmgc**

osprop: Propagating Server Logs

An ObjectStore / Single utility that performs a Server checkpoint.

Syntax

osprop [-f] *server-log-name ...*

Options

-f Instructs **osprop** to ignore errors.

Description

ObjectStore / Single

The **osprop** utility ensures that committed data in the Server logs is propagated to the affected databases. **osprop** is meaningful only when 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 done.

It is not usually necessary to run this utility because an ObjectStore / Single application that terminates normally always 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.

Examples

osprop log1 log2 ...

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 where the ObjectStore / Single version of **libos** is used.

osprop -f log3

Propagates committed data in the specified log file if the file exists and is a valid log. Otherwise, the file is ignored.

API

Class: **objectstore**
Method: **propagate_log**

osrecovr: Restoring Databases from Archive Logs

The **osrecovr** utility copies (rolls forward) database modifications from archive log files to the affected databases.

Syntax

osrecovr [*options***] [-f** *backup/log-file***...] [***pathname_translation...***]**

-f *backup/log-file...*
: Specifies an archive log file 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.

pathname_translation...
: Specifies a pair of pathnames. The first pathname in the pair indicates the source of the database as recorded in the archive log or backup image. The second pathname indicates the target, that is, the pathname for the database after it is recovered.

  You can specify zero, one, or more pathname translations. Each pathname 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 MM/DD/YY format. The **osrecovr** utility rolls forward all database changes committed before or on this date. The default is to roll forward to the last snapshot taken.
-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 "**-**" as the recover file name, **osrecovr** reads from standard input. The list contains one file pathname 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.
-t	Displays a list of databases contained in specified archive files.

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

When restoring data from tape, you must use the **osrestore** utility.

Run **osrestore** and then **osrecovr**

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 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, then 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 where 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

Listing archive contents

```
% osrecovr -f /vancouver1/archives/96011216.aaa -t
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
vancouver::/foo.db
vancouver::/dbdir/bar.db
vancouver::/dbdir/foo.db
Closing volume #1 (/vancouver1/archives/96011216.aaa).
%
```

Recovering from a single archive

```
% osrecovr -f /vancouver1/archives/96011216.aaa
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
Target time: Thu Jan 12 17:28:22 1996
Recovered to time Thu Jan 12 16:25:27 1996
Recovered to time Thu Jan 12 16:25:57 1996
Recovered to time Thu Jan 12 16:26:11 1996
Restoring 452 sectors to database "vancouver:/vancouver1/dbdir/foo.db"
Recovered to time Thu Jan 12 16:26:41 1996
Recovered to time Thu Jan 12 16:27:13 1996
Recovered to time Thu Jan 12 16:27:43 1996
Recovered to time Thu Jan 12 16:28:14 1996
Closing volume #1 (/vancouver1/archives/96011216.aaa).
%
```

Recovering back to place

The next example restores all databases to their location and state as of 16:25:27 and January 12, 1996.

```
% osrecovr -f /vancouver1/archives/96011216.aaa -r 16:25:27
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
Target time: Thu Jan 12 16:25:27 1996
Recovered to time Thu Jan 12 16:25:27 1996
Closing volume #1 (/vancouver1/archives/96011216.aaa).
```

Recovering from multiple archive files

```
% cat ./archive_list
/vancouver1/archives/96011216.aaa
/vancouver1/archives/96011216.aab
/vancouver1/archives/96011216.aac
```

```
% osrecovr -t -F ./archive_list
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
vancouver::/foo.db
vancouver::/dbdir/bar.db
vancouver::/dbdir/foo.db
Closing volume #1 (/vancouver1/archives/96011216.aaa).
```

```
% osrecovr -F ./archive_list
```

```
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
Target time: Thu Jan 12 17:27:01 1996
Recovered to time Thu Jan 12 16:25:27 1996
Recovered to time Thu Jan 12 16:25:57 1996
Recovered to time Thu Jan 12 16:26:11 1996
Restoring 452 sectors to database "vancouver:/vancouver1/dbdir/foo.db"
Recovered to time Thu Jan 12 16:26:41 1996
Recovered to time Thu Jan 12 16:27:13 1996
Recovered to time Thu Jan 12 16:27:43 1996
Recovered to time Thu Jan 12 16:28:14 1996
Closing volume #1 (/vancouver1/archives/96011216.aaa).
Auto switching to volume #2 (/vancouver1/archives/96011216.aab).

Recovering from volume #2 (/vancouver1/archives/96011216.aab)...
Recovered to time Thu Jan 12 16:28:21 1996
Recovered to time Thu Jan 12 16:28:35 1996
Recovered to time Thu Jan 12 16:28:37 1996
Recovered to time Thu Jan 12 16:28:38 1996
Recovered to time Thu Jan 12 16:28:40 1996
Recovered to time Thu Jan 12 16:28:41 1996
Recovered to time Thu Jan 12 16:28:49 1996
Recovered to time Thu Jan 12 16:28:55 1996
Recovered to time Thu Jan 12 16:29:01 1996
Recovered to time Thu Jan 12 16:29:06 1996
Recovered to time Thu Jan 12 16:29:12 1996
Recovered to time Thu Jan 12 16:29:17 1996
Recovered to time Thu Jan 12 16:29:23 1996
Recovered to time Thu Jan 12 16:29:28 1996
Recovered to time Thu Jan 12 16:29:34 1996
Recovered to time Thu Jan 12 16:29:39 1996
Recovered to time Thu Jan 12 16:29:43 1996
Recovered to time Thu Jan 12 16:29:44 1996
Recovered to time Thu Jan 12 16:29:49 1996
Recovered to time Thu Jan 12 16:29:55 1996
Recovered to time Thu Jan 12 16:30:01 1996
Closing volume #2 (/vancouver1/archives/96011216.aab).
Auto switching to volume #3 (/vancouver1/archives/96011216.aac).

Recovering from volume #3 (/vancouver1/archives/96011216.aac)...
Recovered to time Thu Jan 12 16:31:04 1996
Recovered to time Thu Jan 12 16:31:06 1996
Closing volume #3 (/vancouver1/archives/96011216.aac).
%
```

Recovering to a date and time

```
% osrecovr -F ./archive_list -D 1/12/96 -r 16:27:43
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
Target time: Thu Jan 12 16:27:43 1996
Recovered to time Thu Jan 12 16:25:27 1996
Recovered to time Thu Jan 12 16:25:57 1996
Recovered to time Thu Jan 12 16:26:11 1996
```

```
Restoring 452 sectors to database
"vancouver:/vancouver1/dbdir/foo.db"Recovered to time Thu Jan 12
16:26:41 1996
Recovered to time Thu Jan 12 16:27:13 1996
Recovered to time Thu Jan 12 16:27:43 1996
Closing volume #1 (/vancouver1/archives/96011216.aaa).
%
```

Recovering to a time today

```
% osrecovr -F ./archive_list -r 16:27:43
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
Target time: Thu Jan 12 16:27:43 1996
Recovered to time Thu Jan 12 16:25:27 1996
Recovered to time Thu Jan 12 16:25:57 1996
Recovered to time Thu Jan 12 16:26:11 1996
Restoring 452 sectors to database "vancouver:/vancouver1/dbdir/foo.db"
Recovered to time Thu Jan 12 16:26:41 1996
Recovered to time Thu Jan 12 16:27:13 1996
Recovered to time Thu Jan 12 16:27:43 1996
Closing volume #1 (/vancouver1/archives/96011216.aaa).
%
```

Recovering a single database

The next example 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/96011216.aaa)...
Target time: Thu Jan 12 16:27:43 1996
Recovered to time Thu Jan 12 16:25:27 1996
Recovered to time Thu Jan 12 16:25:57 1996
Recovered to time Thu Jan 12 16:26:11 1996
Recovered to time Thu Jan 12 16:26:41 1996
Recovered to time Thu Jan 12 16:27:13 1996
Recovered to time Thu Jan 12 16:27:43 1996
Closing volume #1 (/vancouver1/archives/96011216.aaa).
```

```
% osls vancouver::/
bar.db
dbdir/
foo.db
%
```

```
% cat ./archive_list
/vancouver1/archives/96011216.aaa
/vancouver1/archives/96011216.aab
/vancouver1/archives/96011216.aac
```

```
% osrecovr -t -F ./archive_list
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
vancouver::/foo.db
vancouver::/dbdir/bar.db
vancouver::/dbdir/foo.db
Closing volume #1 (/vancouver1/archives/96011216.aaa).
```

%

Examples of Recovery Failures

Nonexistent database

```
% osrecovr -f /vancouver1/archives/96011216.aaa -r 16:25:27 \
vancouver::/asdla.db vancouver::/as.db
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
Closing volume #1 (/vancouver1/archives/96011216.aaa).
Recover failed: Database vancouver::/asdla.db does not exist in this
backup image
%
```

Day not in the archives

```
% osrecovr -F ./archive_list -D 1/11
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
Target time: Wed Jan 11 17:29:08 1996
Closing volume #1 (/vancouver1/archives/96011216.aaa).
%
```

Day/year not in archives

```
% osrecovr -F ./archive_list -D 1/11/95
Recovering from volume #1 (/vancouver1/archives/96011216.aaa)...
Target time: Tue Jan 11 17:29:51 1995
Closing volume #1 (/vancouver1/archives/96011216.aaa).
%
```

API

None.

osreplic: Replicating Databases

The **osreplic** utility replicates and maintains multiple copies of a database.

Syntax

osreplic[-r] [-v] [-x] [-i *interval***] [-p] [-B** *size***] [-I** *import_file***] -a** *archive_rec_file src_path1 dest_path1* **[***src_path2 dest_path2* **...]**

Options

-a *archive_record_file*	Archive. Required. Specifies the archive record file. If the files does not exist, it is created.
-B *size*	Buffer. Controls the amount of transient workspace available to the source server. *size* is a number optionally appended with **k**, **m**, or **g** to indicate kilobytes, megabytes, or gigabytes 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 megabyte. The default value is 1 MB
-i *interval*	Interval. Sets the interval between snapshots. The default is 600 seconds. A copy is made immediately after **osreplic** is initiated and then every *interval* thereafter. Intervals are specified with integer values. Append **m**, **h**, or **d** to a value to indicate minutes, hours, or days, respectively. By default, values are interpreted as seconds. For example, **-i 60** and **-i 1m** both specify an interval of one minute.
-I *import_file*	Import (uppercase I). 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.
-p	Permissions. Sets database ACLs on the replica to match those of the master (rawfs only).

-r	Recursive. Enables recursive processing of rawfs directories.
-v	Verbose. Enables verbose output.
-x	Exclude. Prohibits clients from using the replica until **osreplic** terminates.
-y	Yes. Confirms the restart of the replicator for an existing master / replica pair and bypasses the usual prompt after **osreplic** starts. When using this option, be sure beforehand that the replica is unchanged since the last update.

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, providing a read-only (MVCC) copy of a database that is dynamically updated from the master database.

ObjectStore Release 4 and later databases and rawfs directories can be replicated, as well as ObjectStore Release 4 and later 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 using the **-I** (I) option. The master / replica pair is specified by pathname with the *src_path* and *dest_path* arguments, including the target host if needed. The list of databases cannot be changed once the replicator is started.

src_path and *dest_path* can be a directory path or a database name. You can also use UNC pathnames (on Windows platforms), Server relative pathnames, or local pathnames. However, you cannot use UNC pathnames as destinations.

You must also specify an *archive_record_file* with the **-a** option. The **osreplic** utility uses this information to determine which segments within a database 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 the

data is applied to the replica. All committed user data is replicated.

Database ACLs (Access Control Lists, including owner, group, mode) can also be copied for rawfs databases. However, neither rawfs directory ACLs nor segment-level permissions are copied. Additionally, no file database ACLs are copied.

Operations such as **osrm** are not propagated to the replica.

API

None.

osrestore: Restoring Databases from Backups

The **osrestore** utility copies databases from backup storage locations to your disk or rawfs. Backups must have been created with the **osbackup** utility.

Syntax

osrestore [*options***] -f** *backup-file* **[-f** *backup-file***]... [***pathname_translation***]...**

-f *backup-file*
: 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. Required.

  On UNIX systems, you can specify **-f -** (hyphen) to indicate **stdin**.

pathname_translation...
: Specifies a pair of pathnames separated by a space. The first pathname in the pair indicates the source of the database as recorded in the backup image. The second pathname indicates the target, that is, the pathname for the database after it is restored.

  You can specify zero, one, or more pathname translations. Each pathname 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
: Aborts the restore operation if the utility cannot open the restore device. This raises an exception that indicates the problem.

  The default is that if the restore utility fails to open the device, it displays a message and waits for you to correct the problem.

  Examples of failure to open the device are having a write-protected tape or no tape loaded.

-b *blocking-factor*	Specifies a blocking factor to use for tape input and output. This parameter 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.
-n	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.
-O	Restores the database image specified with the **-f** flag and then exits. There is no prompt for additional volumes.
-p	The **-p** (permissions) option causes **osrestore** to restore database ACLs for the rawfs stored in the archive log file for the database being restored.
-S *exec_command_name*	Specifies the pathname 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. Note that this option is an uppercase **S**.
-t	Displays a list of databases in the backup image.

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

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. Here is a sample command line for doing this. In this example, **backup.img** contains **foo::/db**. The pathname translation does the job in one step.

```
osrestore -f backup.img foo::/db foo::/restore.db
```

Windows to UNIX pathname translation example

You must specify a pathname translation when you restore or recover data on an architecture that is different from the architecture on which you are restoring the data. For example, here is a Windows NT to UNIX pathname translation. The backup image being restored is **/tmp/my.img**. The interaction is on a UNIX system. You do not need to do anything special when you make the backup on the Windows NT 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 NT 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 back slashes (\) to 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 pathname 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 1995

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 if 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.

Listing databases in backup image

This 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
```

This 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**.

Copying backups to new Servers

Restore 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"
```

Changing Servers and directories

Restore 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"
```

Restoring a single database

Restore 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"
```

Restoring to source with one exception

Restore everything in the **/test** directory on Server **eudyp** to its original location, except **data1.odb**, which gets restored in the **/example** directory on Server **cleopat**.

```
% 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"
```

In this example, the order of the pathname translations is important. Specify specific pathnames before you specify directories that include those pathnames.

Restoring to source

Restore 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"
```

Restoring all to a local directory

Restore 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"

API None.

osrm: Removing Databases and Rawfs Links

The **osrm** utility removes databases and rawfs links from Servers.

Syntax

osrm [-f][i][r] *pathname*...

pathname... Specifies the file or rawfs database or directory, or rawfs link, that you want to remove. You can specify one or more. 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. On OS/2, this option works only on rawfs directories.

Description

To remove a database, you must have write permission to its directory, but you do not need 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, then 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.

The **osrm** utility can perform wildcard processing using regular expression wildcards *****, **?**, **{}**, and **[]**.

For file databases, the **osrm** utility calls the native remove utility.

UNIX

When operating on a rawfs database, you must enclose the wildcard in quotation marks ("") or precede it with a back slash (\) to keep the shell from interpreting wildcards.

API

Class: **os_dbutil**
Method: **remove**

osrmdir: Removing a Rawfs Directory

The **osrmdir** utility removes a directory from the rawfs.

Syntax

osrmdir *directory*

directory	Specifies the pathname of the directory that you want to remove from the rawfs.

Description

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 can also use **osrmdir** to remove a local directory. When you specify a local directory, you cannot specify a remote file-Server host in the pathname 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 pathname.

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 back slash (****) to keep the shell from misinterpreting the asterisk as a shell wildcard.

API

Class: **os_dbutil**
Method: **rmdir**

osscheq: Comparing Schemas

The **osscheq** utility compares two schemas.

Syntax

osscheq [-quiet] *db1 db2*

db1 *db2*	Specifies the pathnames 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.
-quiet	Returns a value of **0** if the databases are compatible. Returns 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.

Example

For example, 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)
```

Comparison technique

The comparison technique depends on the types of schemas being compared. When comparing compilation or application schemas, ObjectStore uses the technique used by the schema generator when building compilation or application schemas. When one of the schemas being compared is a database schema, the comparison technique is the same as that used to validate an application when it accesses a database.

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.

API

None.

osserver: Starting the Server

The **osserver** utility starts the Server. Starting the Server varies from platform to platform. Look for details in the chapter for your platform.

Syntax

osserver *options*

Options

Ordinarily, you use Server parameters to control how the Server functions. However, you can also specify options when you execute the **osserver** utility.

-c Checkpoint. Forces all data to be propagated from the log to the database. The Server does not start after this checkpoint.

-d *int* Starts the Server in debug mode. Specify an integer from 1 through 50. The larger the number, the more information ObjectStore provides. You can also specify the **-F** option so that ObjectStore displays the information on the screen.

ObjectStore copies debug output to the standard Server output file, unless you redirect it to another file.

-F Foreground. Runs the Server process in the foreground. This reverses the normal behavior, where the Server runs as a background process. This option is not available on Windows.

-i Initializes the Server log file and the rawfs, if you have one, with a confirmation prompt. Use with caution.

-I (Uppercase I) Initializes the Server log file, and the rawfs if you have one, *without* a confirmation prompt. Use with extreme caution.

	-p *pathname*	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. This option is not available on Windows.
	-v	Displays Server parameter values at start-up.
API	None.	

ossetasp: Patching Executable with Application Schema Pathname

Syntax

The **ossetasp** utility patches an executable so that it looks for its application schema in a database that you specify.

ossetasp -p *executable*
ossetasp *executable database*

-p	Instructs **ossetasp** to display the pathname of the specified executable's application schema database. Do not specify *database* in the command line when you include **-p**.
executable	Specifies the pathname of an executable. On Windows systems, this can also be the pathname of a DLL.
database	Specifies the pathname of an application schema database. ObjectStore patches the specified executable so it uses this application schema.

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 always be local to the Server.

After you copy or move an application to another machine, you must patch the executable so that it can find the application schema database. Run the **ossetasp** utility with the absolute pathname of the application schema database. Be sure to specify the name of the Server host.

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 281.

Windows

On Windows systems, you can perform the **ossetasp** utility on any EXE or DLL that contains schema (that is, that has a schema object file produced by **ossg** linked into it).

Restrictions

This utility is available on all platforms except OS/2. On OS/2, as well as on all other platforms, you can use the API.

API

Class: **objectstore**
Methods: **get_application_schema_pathname** and **set_application_schema_pathname**

ossetrsp: Setting a Remote Schema Pathname

The **ossetrsp** utility specifies a new pathname for the schema associated with a particular database.

Syntax

ossetrsp {-p | *schema_db_path***}** *db_path*

-p	Displays the pathname 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 some other database.
schema_db_path	Specifies a new pathname for the schema database used by the database specified by *db_path*.
db_path	Specifies the pathname of a database whose schema database pathname you want to either change or display.

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 pathname.

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 pathname.

You cannot associate an entirely new schema database with the main database. You can only change the pathname of the original schema database by moving or copying the original schema database.

API

Class: **os_database**
Methods: **get_schema_database** and **set_schema_database**

ossevol: Evolving Schemas

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 *ObjectStore Advanced C++ API User Guide*, Chapter 9, Advanced Schema Evolution.

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.

Syntax

ossevol *workdb schemadb evolvedb* **...** **[***keyword-options***]**

Options

-task_list *filename*
Specifies that the **ossevol** utility should produce a task list and place it in the file specified by *filename*. Use "**-**" for **stdout**. When you specify this option, ObjectStore does not perform schema evolution.

The task list consists of pseudofunction definitions that indicate how 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.

-classes_to_be_removed *class-name(s)*
Specifies the name of the classes to be removed.

-classes_to_be_recycled *class-name(s)*
Specifies the name of the classes whose storage space can be reused. By default, the storage associated with all classes is recycled.

-drop_obsolete_indexes { *yes* | *no* }
Specifies whether or not obsolete indexes encountered in the course of the evolution should be dropped. The default is **no**, which means that they are not dropped.

-local_references_are_db_relative
Specifies that all local references are relative to the database in which they are encountered. The default is **no**.

-resolve_ambiguous_void_pointers
Resolves ambiguous void pointers to the outermost enclosing collocated object. The default is **no**.

-upgrade_vector_headers

Upgrades the representation of vector objects in the evolved database to a format that allows them to be accessed by clients built by different types of compilers.

You do not need to convert vector objects if the database will be accessed only by applications that were compiled with the same type of compiler. In other words, this option is for databases being used in an environment that includes multiple types of compilers. It is also useful if you are switching from **OSCC** (**cfront**), no longer supported in ObjectStore Release 5.0, to a native compiler that uses vector headers, such as SGI C++.

Use this option only with databases that meet at least one of these conditions:

- Created before ObjectStore Release 4.0
- Built by applications compiled with a **cfront** or **cfront**-derived compiler

You do not need to use this option if you only intend to access the schema from applications that were compiled with **cfront**, since cfront does not need vector headers.

You also need this option on SGI machines when you are converting from ObjectStore 3.*x* to ObjectStore 4.0.

-explanation_level *n*

A number from **1** to **3**; primarily an internal debugging aid.

OS/2 and AIX

It is possible to run the **ossevol** utility on

- An OS/2 machine and evolve a database that was created on AIX
- An AIX machine and evolve a database that was created on OS/2

Description

When you specify two or more classes for an option, separate the class names with a space.

Changes can include

You can use the **ossevol** utility to evolve the following changes:

- Adding/deleting the first/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
- Illegal 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 why the **ossevol** utility might not be able to perform the evolution.

- Deleting obsolete classes / instances. Deleting an object might require modification of data belonging to another object, such as a counter.
- Adding / 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 / 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.

Evolution not required

These changes do not require schema evolution:

- Adding a new isolated class (Isolated means that it is not a superclass or a subclass of another class. When a new class is isolated, its addition does not cause objects already in the database to change.)
- Adding an isolated derived class
- Changing member access (private / protected / public)
- Changing integer data member to / from signed / unsigned
- Changing data member to / from **const** / non-**const**
- Adding / deleting static / persistent data members

Except on Windows NT, the following two changes do not require schema evolution. On Windows NT, these two changes require schema evolution in some cases. You receive a schema validation message when you run the schema generator and schema evolution is required.

- Adding / deleting nonvirtual member functions
- Adding / deleting virtual member functions (other than first or last)

Transformer function required

These changes require application-specific transformer functions:

- Renaming a data member
- Initializing instances of a new data member from data members of old or existing instances
- Changing a data member to / from being indexable
- Adjusting particular local references
- Adding a data member that is a C++ reference or a **const**
- Moving data members to / from a derived or base class
- Reinitialization of user-defined structures containing values that are addresses of evolved instances

Schema protection

When developing an application, if you are 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."
```

When deploying an application, if your end users need to use the **ossevol** utility on protected schema databases, you must wrap the utility in an application. This application must use the API to provide the key before using the **os_dbutil** class to call the utility. End users need not know anything about the key. For information about wrapping your application around an ObjectStore utility, see the class **os_dbutil** in the *ObjectStore C++ API Reference*.

API

For complete information about schema evolution, see *ObjectStore Advanced C++ API User Guide*.

ossg: Generating Schemas

The **ossg** utility is the ObjectStore schema generator. See *ObjectStore Building C++ Interface Applications* for complete information about how to use **ossg**. You must have a client development license and a Server development license to use this utility.

Syntax

Kind of Schema	*Syntax for* **ossg** *Command*
Application	**ossg** [*compilation_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**}] [{ **-sfbp** | **-pfb** }] [**-store_member_functions**] [**-weak_symbols**] {**-assf** *app_schema_source_file* or **-asof** *app_schema_object_file***.obj**} **-asdb** *app_schema_database schema_source_file* [*lib_schema***.ldb** ...] On OS/2, you must also specify **-cd** *class_definition_file*.
Library	**ossg** [*compilation_options*] [*neutralizer_options*] [**-cpp_fixup**] [**-mrscp**] [**-no_default_includes**] [{ **-sfbp** | **-pfb** }] [**-store_member_functions**] [**-use_cf20_name_mangling** | **-use_cf30_name_mangling**] **-lsdb** *lib_schema***.ldb** *schema_source_file*
Compilation	**ossg** [*compilation_options*] [*neutralizer_options*] [**-cpp_fixup**] [**-mrscp**] [**-no_default_includes**] [{ **-sfbp** | **-pfb** }] [**-store_member_functions**] **-csdb** *comp_schema.***cdb** *schema_source_file*

Examples

See *ObjectStore Building C++ Interface Applications*.

Options

compilation_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 OS/2 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.

On OS/2, when you specify an option that takes an argument do not put a space between the option and the argument.

Optional.

neutralizer_options

Include any of the options described in -arch setn on page 245. These options allow you to neutralize a schema for a heterogeneous application. You can include them in any order.

Optional. The default is that neutralization is not done.

-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.

Optional. 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 Object Design support representatives because it allows the problem to be reproduced by Object Design 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 only specify 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 and that 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 pathname with **final_asdb**.

Optional. The default is that the schema generator writes the pathname that you specify **-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 this 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**, then the application schema does not include any types. You must mark at least one type for there to be any reachable types.

Optional. The default is that only marked classes are persistently allocatable and accessible.

See also *ObjectStore Building C++ Interface Applications*, Determining the Types in a Schema.

-mrscp or **-make_reachable_source_classes_persistent**

Causes every class that is both

- Defined in the schema source file
- 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 which types you plan to persistently allocate. 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.

Optional. The default is that only marked classes are persistently allocatable and accessible.

See also *ObjectStore Building C++ Interface Applications*, Determining the Types in a Schema.

-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 in to 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 and OS/2: **/I$(OS_ROOTDIR)\include**

The **-I-** option is the letter I as in *Include.* Specifying **-I-** is the same as specifying **-no_default_includes**.

Optional. The default is that the preprocessor checks default directories for included files.

-no_weak_symbols

Disables mechanisms that suppress notification about missing vftbls and discriminants. This option allows you to check whether any vtbl or discriminant function 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 which 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.

If, in releases prior to 5, you used **os_do_link** with the **-link_resolve_vtbls_and_disc** option, you can now specify **-no_weak_symbols** to perform the same function.

This default option specifies that the schema generator notify you about missing vftbls and discriminants. To change this behavior, specify the option **-weak_symbols**.

-pfb or **-parse_function_body**	Causes **ossg** to parse the code in function bodies. This option ensures that any types that are marked inside a function are parsed by ossg. If you do not explicitly use this option and you have any types marked inside functions, an error is reported. See ossg Troubleshooting Tips for further information. Optional. The default is that the **-sfbp** option is in effect.
-rtdp or **-runtime_dispatch** {**minimal** | **derived** | **full** | **maximal**}	Specifies the set of classes for which the schema generator makes vftbls and discriminant functions 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. In previous ObjectStore releases, this was the default. If your application used an earlier release of ObjectStore and you do not specify this option, you might need to mark classes that you did not previously mark. Optional. The default is **derived**.
-sfbp or **-skip_function_body_parsing**	Default. Specifies that code within function bodies is not parsed.

-store_member_functions

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 is useful when you intend to use the 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 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 the **-store_member_functions** option when you generate the application schema.

Optional. 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 vftbls and discriminants. This option overrides the default behavior described at **-no_weak_symbols** on page 241.

-assf *app_schema_source_file* or **-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**. For all compilers except Visual C++, the schema generator produces a source file that you must compile. When you use Visual C++, the schema generator directly produces the object file.

Required when generating an application schema. No default.

-asdb *app_schema_database*

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 pathname must be local to a host running an ObjectStore Server.

The pathname 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.

Required when generating an application schema. No default.

-csdb *comp_schema***.cdb**

Specifies the pathname of the compilation schema database to be generated by **ossg**. Object Design recommends, but does not require, that the pathname end in **.cdb**.

This pathname must be local to a host running an ObjectStore Server.

Required when generating a compilation schema. No default.

-lsdb *lib_schema***.ldb**

Specifies the pathname of the library schema database to be generated by **ossg**. The pathname must end in **.ldb**.

This pathname must be local to a host running an ObjectStore Server.

Required when generating a library schema. No default.

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.

Almost always required. 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.

lib_schema.**ldb ...**

Specifies the pathname of a library schema database. The name must end in **.ldb**. This can be an ObjectStore-provided library schema or a library schema that you 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.

Optional. The default is that library schemas are not included.

-cd
OS/2 only

On OS/2 platforms, when you invoke **ossg,** you must specify the **-cd** option with the name of the class definition file for the application. The class definition file, also called the schema header file, contains the definitions for all classes that you want in the schema.

Neutralization Options

-arch setn

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.

Required when you are neutralizing schema. No default. You can specify one of the following sets.

set1 Some 32-bit architectures:

HP HP–UX HP C++	RS/6000 AIX C Set ++
IBM VisualAge C++ for OS/2	SGI IRIX SGI C++
Intel Solaris 2 Sun C++	SPARC Solaris 2 Sun C++
Intel Windows NT Visual C++	
Intel Windows 95 Visual C++	

set2 **set1** without some **cfront** architectures:

IBM VisualAge C++ for OS/2	Intel Windows 95 Visual C++
Intel Solaris 2 Sun C++	RS/6000 AIX C Set ++
Intel Windows NT Visual C++	SPARC Solaris 2 Sun C++

set3 Some **cfront** architectures:

HP HP–UX HP C++

set4 Some IBM architectures:

IBM VisualAge C++ for OS/2

set5 **set1** plus AXP Digital UNIX DEC C++ 5.0, with the restriction that your schema cannot contain a data member of type **long**:

AXP Digital UNIX DEC C++ 5.0	Intel Windows 95 Visual C++
HP HP–UX HP C++	RS/6000 AIX C Set ++
IBM VisualAge C++ for OS/2	SGI IRIX SGI C++
Intel Solaris 2 Sun C++	SPARC Solaris 2 Sun C++
Intel Windows NT Visual C++	

set6 **set5** without some **cfront** architectures, and also with the restriction that your schema cannot contain a data member of type **long**:

AXP Digital UNIX DEC C++	Intel Windows 95 Visual C++
IBM VisualAge C++ for OS/2	RS/6000 AIX C Set ++
Intel Solaris 2 Sun C++	SPARC Solaris 2 Sun C++
Intel Windows NT Visual C++	

-neutral_info_output *filename* or **-nout** *filename*

Indicates the name of the file to which neutralization instructions are directed.

Optional. The default is that the schema generator sends output to **stderr**.

-noreorg or **-nor**

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.

Optional. 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. Optional. The default is **-padc**.
-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 *ObjectStore Building C++ Interface Applications*, Listing Nondefault Object Layout Compiler Options in Chapter 5, for details about the content of the option file. Optional. No default.
-show_difference or **-showd** **-show_whole** or **-showw**	Indicates the description level of the schema neutralization instructions. Optional. The default is **-show_whole**.

API None.

ossize: Displaying Database Size

The **ossize** utility displays the size of the specified database and the sizes of its segments.

Syntax

ossize [*options***] *pathname*]**

pathname	Specifies the file or rawfs database whose size you want to display.

Options

-a	Displays the total length of the information segment immediately after the length of the data segment.
-A	Displays access control information.
-c	Displays the type contents for each segment.
-C	Displays the type contents for the entire database.
-f	Displays information about the location of all free blocks of storage in a segment.
-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	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** option, described below.
-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.

-w *workspace-name*	Runs **ossize** with the current workspace set to *workspace-name*, which must be the name of a workspace stored in the specified database. This allows you to examine the size (and contents, with **-c** (**/SEGMENT**) and **-f** (**/FREE**)) of a particular version of the database. If you do not provide this argument, the transient workspace is used as the current workspace (that is, the usual default). If there is a segment that is not known by the current workspace, **ossize** displays Error: there is no version of this segment in this work space**.**
-W	Displays a list of all named workspaces that are stored in the specified database. When specified without other arguments, this option displays only workspace names, with no information about database size.
-0 (zero)	(Zero, not uppercase O) Causes **ossize** to include the internal segment 0 in type summaries. On UNIX and OS/2, this implies **-c** if neither **-c** nor **-C** is set.

Description

The **ossize** utility does not distinguish persistently allocated pointers (that is, pointers to 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 *ObjectStore C++ API Reference*.

PRM format

The **ossize** utility notes the type of PRM entries the database contains. The type can be standard or enhanced. For example:

ossize *<database path>*
Name: /h/kellen/ctdb_1
Size: 74752 bytes (70 Kbytes)
Created: Fri Aug 23 14:59:40 1996

Created by: a SPARC-architecture CPU with 4K pages with Sun C++ 4.x PRMs are in enhanced format
...

ossize *<database path>*
Name: /h/kellen/ctdb_1

Size: 74752 bytes (70 Kbytes)
Created: Fri Aug 23 15:00:53 1996

Created by: a SPARC-architecture CPU with 4K pages with Sun C++ 4.x PRMs are in standard format

Schema protection

When developing an application, if you are 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."

Examples

Rawfs database

```
> ossize lame::/db1
Name: lame::/db1
Size: 44544 bytes (42 Kbytes)
Created: Thu Jan 26 17:19:11 1996
Created by: a SPARC-architecture CPU with 4K pages
There is 1 root:
Name: head   Type: note
There are no external database pointers.
There are no external references.
The schema is local.
There is 1 segment:
Data segment 2:
Size: 512 bytes (1 Kbytes)
>
```

-a

Specifying the **-a** option displays space use information for the information segment. For example:

Info segment usage:

Header/Ovrflw:	512 bytes	(1 * 512)
Tag btree:	512 bytes	(1 * 512)
Tag leaves:	3584 bytes	(7 * 512)
Relocation map:	512 bytes	(1 * 512)
Free Tree:	3584 bytes	(7 * 512)
Hugespace:	2048 bytes	(4 * 512)
Fixed Cluster:	512 bytes	(1 * 512)
String Pool:	2048 bytes	(4 * 512)
Unused:	1024 bytes	(2 * 512)
Total Size:	14336 bytes	(28 * 512)

-o

Specifying the **-o** option displays fixed cluster locations for each segment. For example:

Fixed Offset	Cluster Size
0	4096
0x1000	8192
0x3000	16384
0x7000	32768
0xf000	4096
0x10000	65536
0x20000	8192

Fixed Offset	Cluster Size
0	4096
0x22000	16384
0x26000	32768
0x30000	65536

API

Class: **os_dbutil**
Method: **ossize**

ossvrchkpt: Moving Data Out of the Server Transaction Log

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.

Syntax

ossvrchkpt *hostname*

FAT name

ossvrchk

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:

0	Success.
1	There is an error when passing the command to the Server.
2	The Server is unable to complete the checkpoint.

When needed

Run this utility when you want to

- Ensure that a Server is restarted as fast as possible
- Use operating system file commands such as copy or move on a file database
- Reinstall or upgrade ObjectStore

Example

ossvrchkpt hostess

Data in the Server transaction log on the host called **hostess** is copied to the databases that were modified.

API

Class: **os_dbutil**
Method: **svr_checkpoint**

ossvrclntkill: Disconnecting a Client Thread on a Server

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

ossvrclntkill *hostname* **-h** *client-host* **| -p** *client-pid* **| -n** *client-name* **[-a]**

ossvrclntkill *hostname client-pid*

(The second form is supported for compatibility with earlier releases.)

Options

hostname	Specifies the name of the host of the Server that is connected to the client process being disconnected.
-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.

You can use **ossvrstat** to determine the *client-hostname* and *client-pid.*

When needed

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 when 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.

UNIX

You must specify **-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 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.

Example

ossvrclntkill hostess -h cupcake -a

This disconnects all clients on **cupcake** that are attached to the Server on **hostess**.

API

Class: **os_dbutil**
Method: **svr_client_kill**

ossvrdebug: Setting a Server Debug Trace Level

Syntax

ossvrdebug *hostname n*

Options

hostname	Server host to be debugged.
n	Number that specifies the trace level of the Server.

Description

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. The requested trace output is put into the **/tmp/ostore/oss_out** file (on UNIX) once the Server receives the message.

Example

ossvrdebug kellen 5

Sets the Server debug trace level of the server **kellen** to 5.

API

Class: **os_dbutil**
Method: **ossvrdebug**

ossvrmtr: Displaying Server Resource Information

The **ossvrmtr** utility provides information about resource use for the Server process running on the specified host.

Obsolete

This utility actually calls the **ossvrstat** utility. Any information you can obtain by running **ossvrmtr**, you can also obtain by running **ossvrstat**. The **ossvrmtr** utility will not be supported in future releases.

Syntax

ossvrmtr *hostname*

hostname	Specifies the name of the host of the Server for which you want to display information.

Description

The **ossvrmtr** utility summarizes metering information for total clients and for logs for these intervals:

- The last minute
- The last ten minutes
- The last hour
- Since start-up

You can use the **ossvrstat** command to see the Server information and per-client information.

Example

See the **ossvrstat** example on Messages received on page 262.

API

Class: **os_dbutil**
Method: **svr_stat**

ossvrping: Determining If a Server Is Running

The **ossvrping** utility reports whether or not a Server is running on the specified host.

Syntax

ossvrping [-v] [*hostname* **]**

Options

hostname	Specifies the name of the host on which you want to know whether or not a Server is running.
-v	Indicates that you want more information when a Server is not running on the specified host.

Description

If you are having any problems with a Server, the first thing to do is run **ossvrping** to see if the Server is running.

If you do not specify a host, the default is the local host.

Examples

ossvrping elvis
The ObjectStore Server on host elvis is alive.

API

Class: **os_dbutil**
Method: **svr_ping**

ossvrshtd: Shutting Down the Server

The **ossvrshtd** utility immediately shuts down the Server running on the specified host. This is regardless of whether or not clients are connected to the Server.

Syntax

ossvrshtd [-f] *hostname*

Options

-f	Specifies that shutdown should be immediate. When you do not include this option, ObjectStore prompts you to confirm that you really want to shut down the Server. When you include this option, there is no confirmation prompt.
hostname	Specifies the name of the host of the Server that you want to shut down.

Description

Before shutting down the Server, run the **ossvrstat** utility to determine if there are clients using the Server. If there are, notify them to exit.

Clearing the log

Shutting down the Server automatically propagates everything in the transaction log.

If any clients are connected to the Server when you shut it down, the next time those clients try to contact the Server they receive the message err_broken_server_connection. The client can call **os_server::reconnect** to try to reconnect.

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**.

Windows NT

On Windows NT, you can shut down the Server using the Service Control Manager. Click on the Services icon in the Control Panel or issue the command **net stop "ObjectStore Server R5.0"**.

OS/2

You must be the Administrator to run this utility.

Starting a Server

For instructions for starting a Server, see the chapter in this book for your platform.

Example

```
ossvrshtd hostess
Are you sure that you wish to shut down the server
on host hostess (yes/no) [no]: yes
```

API

Class: **os_dbutil**
Method: **svr_shutdown**

ossvrstat: Displaying Server and Client Information

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***]**

Options

hostname	Specifies the name of the host of the Server for which you want information.
-meters	Displays performance meters for the specified Server.
-clients	Displays the state of each client connected to the specified Server, and shows which clients, if any, are contending for locks.
-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.

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**.

Numbers are relative

The table on the next few pages describes the information that **ossvrstat** supplies in terms of *high* and *low* numbers. This is entirely relative to your application. For *high* and *low* to have meaning, run **ossvrstat** to determine a baseline.

Display all

To display all information, do not include any options. The following table describes the meters that **ossvrstat** provides.

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 how 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, sending data, updating a database, committing a transaction, aborting a transaction, or closing a database. This 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 locked by clientA. When this number is high, it means that an application is often modifying data that other clients also want to modify. This might mean that the program is poorly designed.
Callback sectors	Number of sectors that have been called back in callback messages. This is not necessarily the same as the number of sectors for which locks have actually 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 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 with the lock (clientA) either shared the lock with clientB or relinquished the lock 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, then 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 or not 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 have 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 being written past the current end of a segment. When analysis is concerned with the number of transactions per second, the number of kilobytes written is an important factor.

Meter	*Description*
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.
Readonly commits	Number of committed transactions that the Server determined did not involve 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 changes 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** and is in microseconds. 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 which clients are waiting for locks and which clients currently have those locks.

Meter	*Description*
Message buffer waits	Number of times a message from a client to the Server must wait to use a message buffer. The **Message Buffers** Server parameter specifies how many message buffers the Server uses to communicate with clients. If the number of **Message buffer waits** is high, consider increasing the value specified for **Message Buffers**. See **Message Buffers** on page 86.
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 zero, ObjectStore does not print information for these two meters.
Notifies received	Number of notification messages the Server received from all clients. The Server then 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 zero, 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.
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. For descriptions of the segments in the transaction log, see Log File Terms on page 17. To switch segments, the Server must ensure that all changes that are recorded in the log record segment being switched to have been propagated to the databases. When the Server needs to switch log record segments, if not everything is propagated then 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** on page 84.
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.

Meter	*Description*
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 prior to 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 continually reused.
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.
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. The Server stores uncommitted data in the database when an application tries to write data past the end of the database segment in which it needs to be stored. The **Direct To Segment Threshold** Server parameter controls how far the Server can write past the current end of the database segment before the Server writes the data directly to the database. See **Direct to Segment Threshold** on page 81.

Meter	*Description*
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. 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** on page 84. • 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** on page 82. • 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 how to specify the amount of data moved in one propagation, see **Max Data Propagation Per Propagate** on page 85.

Example

```
ossvrstat kellen
ObjectStore Release 5.0 Database Server
Client/Server protocol version 1.8
Compiled by staff at 97-02-18 17:40:42 in
/h/kellen/1/r5core/obj/sun4/opt/nserver

Allow Shared Communications: Yes
Authentication Required: SYS, DES, Name Password
Cache Manager Ping Time:300
Cache Manager Ping Time In Transaction:300
DB Expiration Time: 5 seconds
Deadlock Victim: Work
Direct To Segment Threshold: 128 sectors (64KB)
Log File:                    /kellen/log_file_DB
Current Log Size43024 sectors (21512KB)
Log Data Segment Growth Increment: 2048 sectors (1MB)
Log Data Segment Initial Size: 2048 sectors (1MB)
```

```
Log Record Segment Buffer Size: 1024 sectors (512KB)
Log Record Segment Growth Increment: 512 sectors (256KB)
Log Record Segment Initial Size: 1024 sectors (512KB)
Max AIO Threads3
Max Connect Memory Usageunlimited
Max Data Propagation Per Propagate: 512 sectors (256KB)
Max Data Propagation Threshold: 8192 sectors (4MB)
Max Memory Usageunlimited
Max Two Phase Delay30
Message Buffer Size: 512 sectors (256KB)
Message Buffers: 4
Notification Retry Time: 60 seconds
Preferred Network Receive Buffer Size16384 bytes
Preferred Network Send Buffer Size16384 bytes
Propagation Sleep Time: 60 seconds
Propagation Buffer Size: 8192 sectors (4MB)

Server Machine Usage:
    User time:58123.6 secs
    System time:3151.1 secs
    Max. Res. Set Size:6639
    Page Reclaims:1400444
    Page Faults:54510
    Swaps:0
    Block Input Operations:20339
    Block Output Operations:387732
    Signals Received:1
    Voluntary Context Switches:775502
    Involuntary Context Switches:645611

Server Meters:
Total since server start up:
    Client Meters:
        1314496 messages received 23575 callback messages sent
        211960 callback sectors     94240 succeeded sectors
        3192253KB read          3691926 KB written
        211351 commits 89572 readonly commits
        19749 aborts 0 two phase transactions
        0   lock timeouts341lock waits (average 7555 us)
        74 deadlocks 0 message buffer waits
        14896notifies sent14938notifies received
    Log Meters:
        219314 log records 1514 record segment switches
        52115 flush data        225902 flush records
        0   KB data             0 KB records
        576954 KB propagated 201924 KB direct
        28302 propagations

Total over past 60 minute(s):
    Client Meters:
        2135 messages received 0 callback messages sent
        0   callback sectors        0 succeeded sectors
```

```
        7416 KB read 2759 KB written
        40 commits 20 readonly commits
        91 aborts 0 two phase transactions
        0  lock timeouts0lock waits
        0  deadlocks 0 message buffer waits
        1843notifies sent1772notifies received
    Log Meters:
        116 log records 10 record segment switches
        14 flush data 187 flush records
        0  KB data 0 KB records
        1799 KB propagated 787 KB direct
        102 propagations

Total over past 10 minute(s):
    Client Meters:
        1056 messages received 0 callback messages sent
        0  callback sectors 0 succeeded sectors
        3708 KB read 1383 KB written
        20 commits 10 readonly commits
        44 aborts 0 two phase transactions
        0  lock timeouts0lock waits
        0  deadlocks 0 message buffer waits
        1843notifies sent1772notifies received
    Log Meters:
        57 log records 5 record segment switches
        7  flush data 73 flush records
        0  KB data 0 KB records
        901 KB propagated 395 KB direct
        51 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 timeouts0lock waits
        0  deadlocks 0 message buffer waits
        0  notifies sent0notifies received
    Log Meters:
        0  log records 0 record segment switches
        0  flush data 1 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 man 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 something like 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()** method. 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 or not a transaction is in progress.

API	Class: **os_dbutil** Method: **svr_stat**

ostest: Testing a Pathname for Specified Conditions

The **ostest** utility indicates whether or not a pathname meets a specified condition.

Syntax

ostest [*option*] *pathname*

pathname	The pathname of a database or directory.

Options

-d	*pathname* is a rawfs directory.
-f	*pathname* is a rawfs database.
-p	*pathname* is a file pathname.
-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*.

Description

You can specify one option when running this utility. The **ostest** utility returns an exit code of

0	When the specified condition is **true**
Nonzero	When the specified condition is **false**

When you specify a file database, you cannot specify a remote file-server host in the pathname 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 pathname.

API

Class: **os_dbutil**
Method: **stat**

osupgprm: Upgrading PRM Formats

Upgrades a database's address-space format.

Syntax

osupgprm *database-name ...*

Description

This utility changes the address space format for a database to use a PRM (persistent relocation map) format derived from deferred address space reservation.

Immediate assignment

Prior to Release 5, address space for a segment was always reserved immediately. Immediate reservation means that the first time any page in a segment is accessed, all of the address space required for that segment, including pointers out of the segment to other segments, is reserved. In some cases, this results in excessive use of address space.

Deferred assignment

Deferred assignment means that the first time that a page in a segment is accessed, the minimal amount of address space required for that page, including pointers out of that page, is reserved. Any new databases created with Release 5 are automatically created using enhanced PRM entries, unless you explicitly specify the standard format.

Upgrade is recommended

In order to take advantage of deferred address assignment, existing databases created from previous releases of ObjectStore should be upgraded to a new enhanced PRM format. Object Design recommends this upgrade in almost all cases. Release 4 clients can only access databases using the standard (old) PRM format.

The choice of immediate as opposed to deferred assignment for a segment is made every time the segment is put in use for the first time in a transaction. The type of assignment must remain constant for the duration of a transaction.

First run osprmgc

Before upgrading a database, use **osprmgc** to conserve currently reserved address space for the database.

Cross-database pointers

Release 5 clients can access databases that use either the standard (pre-release 5) or enhanced PRM format, but the cross-database

pointers must be between databases that use the same PRM format. Specify the database you want to upgrade and its target databases in any order. Target databases are also upgraded to use deferred address space reservation.

osverifydb: Verifying Pointers and References in a Database

The **osverifydb** utility verifies all pointers and references in a database.

Syntax

osverifydb [*options***]** *pathname*

pathname	Specifies a file or rawfs database whose pointers you want to verify.

Options

-all	Verifies all segments including the internal segment 0.
-end_offset *integer*	Specifies the end offset (in bytes) within the segment where verification is done. Defaults to **0,** which starts verifications at the end of the segment.
-ignore_references	Suppresses verification of references.
-illegal_pointer_action {null | ask}	When used with the **-all** option and **null** argument, sets the illegal pointer to null. With the **ask** argument, uses the reference value that is supplied in response to the query.

-info_sector_tag_ verify_opt *option*	Checks that a database created on an SGI machine with a 16 K page size in an ObjectStore release prior to 5.0 can be used by heterogeneous ObjectStore applications. This option can also be used to upgrade such a database for use with heterogeneous applications if needed. Valid *option* values are: **0** - Skips verifying info segment sector tags (default). **1** - Verifies info segment sector tags and reports. whether the database can be used heterogeneously. **2** - Upgrades the database for heterogeneous accessibility. **5** - Causes **osverifydb** to report information for this option only. Other verifications usually performed by **osverifydb** are not made. **6** - Performs an upgrade only. Other verifications usually performed by **osverifydb** are not made.
-L *server-log-name*	When specified, the named file is used for the Server log file. When unspecified, a temporary file is used. This option is only applicable when you are running the utility as an ObjectStore / Single application. If the file already exists, it must be a properly formed Server log.
-n *segment-number*	Verify only the segment specified by *segment-number*.
-nocoll	Suppresses integrity checks that ensure that the ObjectStore collections in the database are valid. Object Design recommends that you use this option only on databases that do not contain collections.
-o	Displays each object in the database using the metaobject protocol.
-start_offset *integer*	Specifies the start offset (in bytes) within the segment where verification is done. Defaults to **0**, which means start verifying at the beginning of the segment.
-tag	Displays the tag value on an error.
-v	Displays the value for each pointer.

-whohas *hex_address* Lists objects that point to the object identified by the pointer.

Description

Verification means

- 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

Reference verification requires that the reference be resolved to an address before it can be verified. This requires additional address space resources. In some cases, the **osverifydb** utility might run out of address space. Turning off reference verification allows verification of a database in such circumstances.

You would not normally include the **-ignore_references** option unless you had already tried to verify the database and verification failed because the utility 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.

Schema protection

When developing an application, if you are 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."

Example

osverifydb -all -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 in the format used by **os_reference::load()**. Here is some sample output from **osverifydb** in such a circumstance:

```
The object at 0x6020000 (</daffy/home/daffy/daffy0/dbs/verifydb1 | 2 |
0>)(type "c1"), contains a pointer at 0x6020000(c1.m1) with the illegal
value 0x1. It points to nonpersistent storage.
Enter replacement pointer value in reference dump format (<database
path | segment number | hex offset>:
```

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 because 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%
```

API

Class: **os_dbutil**

Method: **osverifydb**

osversion: Displaying the ObjectStore Version in Use

The **osversion** utility displays the version of ObjectStore that is in use on your machine.

Syntax

osversion

Examples

SPARCstation

elvis% **osversion**
ObjectStore Release 5.1 for SPARC Solaris 2

Windows

[D\:] **osversion**
ObjectStore Release 5.1 for Windows NT Systems

OS/2

[D\:] **osversion**
ObjectStore Release 5.1 for OS/2

API

Class: **os_dbutil**

Methods: **release_name**
release_major
release_minor
release_maintenance

Also see the file **include/ostore/osreleas.hh**.

Chapter 5
Using Locator Files to Set Up Server-Remote Databases

This chapter provides information for 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

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 that are not stored on an ObjectStore Server host.

Rawfs databases

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 that

- 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.

Caution

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.)

Caution

In accessing 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.

What Are the Steps for Allowing 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* 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 which ObjectStore Server should handle access to which Server-remote databases. Locator files do not need to 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 which Server 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.

Shared **$OS_ROOTDIR**

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 file with either a client environment variable or an API call. These are discussed in Overriding the Default Locator File on page 299.

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 when one Server handles access to a database, it can prevent concurrent access by other ObjectStore Servers. This is discussed further in When Multiple Servers Can Concurrently Access a Database on page 300.

Prototype and study

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 if 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) where the database resides
- The pathname of a database or group of databases
- Which ObjectStore Server controls access to the specified database
- How 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

Example

This 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 below.

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 pathname 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 pathname 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 pathname 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 pathname that is canonical for the file server host **oak**, that is, **c:\oak1\test1\test1.db**. This translation is the pathname by which the database is known to the file server, and it is used as input to the locator file.

The rule above applies in this case, because the file server host has the specified name (**oak**) and the input pathname **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 just 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 go 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 | unc }**

where *host-name-pattern* is a character string pattern that specifies a set of host names. The rules about how character string patterns are written and used are discussed in Using Character String Patterns in Locator Files on page 294. 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 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 290.

The following table shows how a host declaration determines an associated standard delimiter:

Operating System	*Delimiter*
unix	**/**
pc	****
unc	****

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***]**

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 pathname by which that database is known to that ObjectStore Server

If ObjectStore does not find a rule that applies to a database, then 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 where 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 294.

If a database specified in an application resides on a file server host whose name has this form, then the rule containing this statement might apply to the database (see **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 pathname 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 294.

If the name by which a database is known to the file server matches *pathname-pattern*, then the rule containing this statement applies to the database, provided 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-names*:

- **@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 how 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 pathname. This is the pathname 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 pathname (**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 ("x"). If you want the quotation marks to be interpreted literally, use two consecutive quotation marks (""x"").

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 which pattern is matched, but 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 “**.**”.

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 288.

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. For example, if all machines using ObjectStore use a PC syntax for file pathnames, then a single **HOST** declaration that covers them all would be

HOST declaration example

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 do not need to modify the locator file when you add a new PC host to the network.

Directory specification example

Here is another example. Suppose 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 to know when writing character string patterns. An understanding of the following terms is needed to correctly apply the rules.

Definitions of terms

The *target* is character string input. For example, the pathname 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

Here 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 metacharacters

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 metacharacters

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.
[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.

Metacharacter	*Description*
	When a caret (^) is the first character after the left bracket, the sense of the match is reversed. In other words, 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 some caution is advised. It is safe to assume that **[0-9]** means the ten digits on any ASCII system.
(and **)**	An open parenthesis starts a grouping. A close parenthesis ends a grouping. Parentheses group multiple characters so that the software can treat them as a unit. This is useful if sequences might repeat in the target, as shown in the examples above for asterisk and plus. Parentheses are also especially useful to delineate a grouping when you are doing string replacements. See the discussion of the **REPLACE** command in REPLACE syntax on page 291.
$	A dollar sign is an escape character. The purpose of the escape character is to provide a work around when 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 **+**.

Metacharacter	*Description*
	You must escape the following characters when they are not used as metacharacters: **. * + [ ] ( ) $ ^** . The backslash (****) character has no meaning as a metacharacter; you do not need to 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 111.
" "	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 and OS/2	**%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 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.

Or, you can 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 / 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 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

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. A mistake in a locator file can cause unintentional concurrent access of this sort.

Caution

The recommended mode of operation is

- Database-level locking is on (the default).
- Each database that is ever opened for read / 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 allows access to databases on a UNIX server from a Windows ObjectStore Server. The ObjectStore Server is on the local host.

UNIX file host

To access Server-remote databases on a UNIX host from a Windows ObjectStore Server, you can modify the locator file as shown.

```
HOST @LOCALHOST pc
HOST andover unix

FILE_HOST andover
FILE_PATHNAME /h/andover/1/.+
SERVER_HOST @localhost
REPLACE /h/andover/1 u:
REPLACE_DELIMITERS

FILE_PATHNAME .*
SERVER_HOST @LOCALHOST

END
```

Remember that case is not significant for locator file keywords, so **@localhost** is acceptable. The **REPLACE** statement is important because ObjectStore must translate the Windows pathnames to UNIX pathnames. The **REPLACE_DELIMITERS** statement is also necessary for correct pathname translation.

Wildcards in host declarations

Suppose you want to set up access to databases on a large number of Server-remote hosts. This example shows an easy way to do it.

```
HOST andover unix
HOST .* pc

FILE_HOST andover
FILE_PATHNAME /h/andover/1/.+
SERVER_HOST @localhost
REPLACE /h/andover/1 u:
REPLACE_DELIMITERS
END
FILE_HOST .*

FILE_PATHNAME .*
SERVER_HOST @LOCALHOST
END
```

Use a wildcard to specify the largest group of hosts that are the same type. This works because ObjectStore searches locator rules

sequentially and uses the first rule that applies. It is not an error for multiple rules to apply.

NFS-mounted **$OS_ROOTDIR** with same architectures

Here 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**, is the ObjectStore client.

Here 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 jeep
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 where one of the machines mounts **$OS_ROOTDIR** from another machine. The machine named **screamer** is an HP with its own **$OS_ROOTDIR**, but without a local Server. The machine **towanda** is a Sun 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 another Server host

The next locator file shows how 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 this 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

Here 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 like 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

Here is another example of an incorrect locator file:

```
#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 like 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)
```

Limitations When Using NFS to Access Remote Databases

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 below describes NFS problems that can occur when you use locator files. These limitations are because of known NFS problems and do not necessarily apply when 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. Object Design recommends not using the automounter. However, even hard mount points can become stale.	None. Object Design recommends that you do not use the automounter.
Server crash. The **oss_out** file contains references to permission problems.	Poorly configured NFS. You must configure NFS such that **anon**=**0**. When an application creates or opens a database, the Server acts on behalf of the user who is requesting the open/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, then 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 (where the database resides).	Configure NFS with **anon**=**0**.

Symptom	*Reason*	*Workaround*
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, then NFS maintains a copy of the database so that 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), then 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.
Server hang (wait state).	The Server requires the lock manager to be running on the machine that the Server is running on. 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. Once 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).	Slow response from NFS. If, for any reason, NFS is slow to respond to requests for service, the Server must wait for those responses, just like any other client of NFS.	None.

Troubleshooting

If your locator file is not functioning the way you intend, first carefully check the locator file for these 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 accurately specify any **REPLACE** statements?

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

Object Design provides a number of facilities for increased levels of reliability. In descending order of reliability the tools and capabilities are

- Warm failover
- Asynchronous replication
- **osbackup** (See Chapter 1, Overview of Managing ObjectStore, General Backup Practices on page 41)
- **osarchiv** (See Chapter 1, Overview of Managing ObjectStore, Archive Logging on page 43)

This chapter discusses

Warm failover addresses the issue of local availability of data in the face of a system hardware or software failure. It consists of two sets of failover features — Server failover features and ObjectStore APIs useful to an ObjectStore client application writer.

Asynchronous replication addresses the issue of wide-area availability by providing a method for replicating data to another system, potentially at another geographic location.

Warm Failover

ObjectStore failover features help solve the local availability problem in cases of system hardware or software failure. The ObjectStore Release 5.1 failover system provides a two-part set of failover features. The Server-side failover system enables a database administrator to set up a configuration that provides the Server failover system. Client-side failover capabilities consist of a new set of ObjectStore APIs that enable client application writers to take advantage of this facility.

Failover provides nonstop service of client applications so that a single Server failure does not affect a running application. It does this by shifting processing to a secondary Server system in the event of a failure of the primary system.

A failover system is made up of two ObjectStore Servers, running on different machines, that share a disk. If the primary Objectstore Server fails, the client process continues operation by connecting to the secondary Server. Service is uninterrupted and you can continue to access and modify the database.

The primary Server process accepts connections from an ObjectStore client. The secondary Server does not accept connections from clients, but waits offline, ready to take over should the primary Server fail.

Restrictions

Failover can be used with rawfs partition databases. File databases are not accessible by a failover Server (nor are two-phase commits allowed).

Warm failover alone cannot guarantee 100% fault tolerance as the shared disk can crash, the network can fail, and so on.

Each of the ObjectStore Servers that implements a failover Server must have a rawfs and log file on a disk that is shared by the two machines upon which the Servers are running. They must also be running on the same software architecture.

The ObjectStore client run time, through the use of the new APIs, locates the on-line Server of the failover Server using a locator file that is local to the client. The locator file declares configuration information for a failover Server.

Configuration of the Shared Disk

The configuration depends on a shared, reliable disk (such as RAID 5) 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 (for Digital or HP systems, for example).

Server Configuration

Each Server in the failover Server pair has a parameter file. The parameter file must specify two things, the heartbeat time and the physical rawfs partitions. The Server parameter files for a failover Server pair must share the same physical partition and log file. The log file must be in the rawfs, not in a file.

You must set the **Failover Heartbeat Time** parameter in the Server parameter file and restart the Server with the **-upgradeRAWFS** switch set. To reconfigure without failover, remove the parameter from the Server parameter file and restart the Server with the **-upgradeRAWFS** switch. The next paragraph describes this parameter.

Failover Heartbeat Time

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 **Failover Heartbeat Time** for the secondary Server to recognize the failure, initialize, and take over.

Example

Partition0: partition /dev/rdsk/c0t0d0s3
Failover Heartbeat Time: 2

How a Failover Database Is Located by the Client Application

The locator file provides a way of defining pairs of ObjectStore Servers that compose a failover Server. If a Server is not available and the locator file has an alternative Server, the client tries this Server to see if it can provide access to the rawfs database.

Logical Server Names

There are two physical Server processes responsible for serving one logical rawfs. Only one of the Server hosts should ever be

recorded in the database pathnames referring to databases maintained on a failover Server. The Server host that is recorded as part of the rawfs database path is referred to as the *logical Server host name*.

The ObjectStore client application should always use the logical Server host name when manipulating a database on a failover Server. This is because, when an application creates a database on the failover Server and the secondary Server is on line, the pathname of the database includes the logical Server host name, *not* the name of the secondary Server that created the database. If the application uses the nonlogical Server name in a pathname, the exception err_not_supported is raised.

Declaring a Failover Server in a Locator File

See Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 281, for general information about locator files.

When using locator files to implement failover Servers, be certain that the locator file is in a file that is local to the client application so NFS is not involved in a failover situation.

Do not add failover Server declarations to pre-5.0 ObjectStore client system locator files. ObjectStore clients will fail when trying to parse a locator file containing failover Server declarations.

Locator file syntax has been extended to support failover Servers, as shown in the following failover Server clause:

```
FAILOVER_SERVER server_host_1
    ALTERNATIVE_SERVER server_host_2
    RECONNECT_TIMEOUT integer # in seconds
    RECONNECT_RETRY_INTERVAL integer # in seconds
END
```

server_host_1 is also the logical Server host name that is recorded in all database pathnames to databases of this failover Server. *server_host_2* specifies the backup Server of a failover Server pair. Both values are required.

RECONNECT_TIMEOUT is the maximum amount of time that a client will attempt to reconnect to a failover Server before raising the exception err_broken_replicated_server_connection.

RECONNECT_RETRY_INTERVAL specifies how often the two failover Servers are pinged during the **RECONNECT_TIMEOUT**. **RECONNECT_RETRY_INTERVAL** should be less than or equal to **RECONNECT_RETRY_TIMEOUT**.

Also, **RECONNECT_RETRY_INTERVAL** cannot be zero if **RECONNECT_TIMEOUT** is nonzero. The exception err_locator_syntax is raised if these constraints are violated.

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 method **objectstore::set_locator_file()** is called. Then, the locator file is reread the next time the ObjectStore run time uses its contents.

Notes for Writing Client Applications

If an application is written using lexical transactions (that is, the transaction macros), the retry mechanism in the client is done automatically.

This means that client programs written using lexical transactions do not need to be modified to take advantage of the failover feature.

The Failover API

Failover features available to client application writers include a set of APIs that can be incorporated into client applications. These allow applications to access environment information for failover.

Use of these interfaces is not required. They include the **os_failover_server** class and functions in other classes, all described here.

objectstore::get_locator_file()

static char* get_locator_file() ;

Returns a string representing the locator file. If the first character of the string is a white-space character or #, the string is the contents of the file rather than a file name.

The caller should delete the returned value.

os_server::get_host_name()

char* get_host_name();

For failover Servers, this function returns the logical failover Server host name. Note that the logical Server name is not always identical to the Server name for the machine providing access to the database. The caller should delete the returned value. See **os_failover_server::get_online_server()**.

os_server::is_failover()

os_boolean is_failover() const;

Returns **true** if and only if the Server is also a failover Server.

This method is used to identify the **os_failover_server** in the list of Servers returned by **objectstore::get_all_servers()**.

The os_failover_server Class

class os_failover_server : public os_server

This class is derived from **os_server**.

The types **os_int32** and **os_boolean**, used throughout this manual, are each defined as a signed 32-bit integer type. The type **os_unsigned_int32** is defined as an unsigned 32-bit integer type.

Programs using this class must include **<ostore/ostore.hh>**, followed by **<ostore/coll.hh>** (if ObjectStore collections are used).

os_failover_server::get_logical_server_hostname()

char* get_logical_server_hostname() const;

Returns the logical name of a failover Server. A failover Server should always be referred to by its logical Server name.

The caller should delete the returned value.

os_failover_server::get_online_server_hostname()

char* get_online_server_hostname() const;

Returns the Server that the client is currently connected to, either the logical Server, alternative Server, or the empty string if there is no connection.

The caller should delete the returned value.

os_failover_server::get_reconnect_retry_interval()

os_unsigned_int32 get_reconnect_retry_interval() const;

Returns the frequency with which to ping both Servers composing a failover Server pair while attempting to reconnect to them.

os_failover_server::get_reconnect_timeout()

os_unsigned_int32 get_reconnect_timeout() const;

Returns the maximum amount of time that a client application will attempt to reconnect to a broken failover Server connection.

After this amount of time passes, the following exception is raised:

err_broken_failover_server_connection

os_failover_server::set_reconnect_timeout_and_interval()

os_boolean set_reconnect_timeout_and_interval(
os_unsigned_int32 total_timeout_secs,
os_unsigned_int32 interval_secs);

Sets the total amount of time to try to reconnect a broken connection to a failover Server. The **interval_secs** argument is used to control how frequently the Servers of a failover Server pair are pinged to see if they are available.

Returns **true** if the reconnect timeout has been reset with the specified parameters.

If the parameters are invalid, the function returns the value false and does not change the **reconnect_timeout** or **reconnect_retry_interval**.

Invalid parameters are those for which

- **interval_secs** is greater than **timeout_secs**.
- **interval_secs** is set to **0** when **timeout_secs** is nonzero.

Exceptions and Error Messages for Failover

This section presents run-time exception cases that can occur with failover.

err_failover_server_refused_connection

Raised when the initial connection to a failover Server pair cannot be made.

err_broken_failover_server_connection

Raised 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

Raised 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 raised within them.

err_not_supported

Raised when the alternative Server name is used directly to reference a database. An Objectstore application should only reference the logical Server name of a failover Server pair.

Also raised when **os_dbutil::ping_failover_server** is called and the locator file does not declare *hostname* as a failover Server.

err_conflicting_failover_configuration

Raised by **os_dbutil::ping_failover_server()** if both Servers composing the failover Server pair are alive or if a server stat of the on-line Server indicates that it is not a failover Server.

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 dynamically updated from the master database.

ObjectStore Release 4 and later databases and rawfs directories can be replicated, as well as all ObjectStore Release 4 file databases. Native file system directories cannot be replicated.

See osreplic: Replicating Databases on page 213 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 along with this chapter.

The topics discussed in this chapter include

Database and Executable Pathnames

You specify a file database with an operating system pathname. For example:

```
os_database::open("/usr3/fauntleroy/my_file_db")
```

You can also specify a relative pathname.

ObjectStore treats links in the usual manner.

Automount pathnames are acceptable. If you are using **automount**, the pathname you specify cannot include the automount prefix, usually **/tmp_mnt**. Referring to the file with this prefix does not cause **automount** to keep it mounted and can cause the file to appear 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 281), you must store file databases on a host that is running an ObjectStore Server. ObjectStore determines which Server to use based on the NFS mountings of the client that creates the database.

Colons in file pathnames are interpreted as alphabetic characters if a slash character precedes the colon in the pathname. For example:

/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.

File Name Expansion

When you specify a rawfs database name, ObjectStore commands use wildcard (*****, **?**, **{}**, and **[]**) name expansion.

You must insert a back slash (\ ) as an escape character immediately before a wildcard character. For example:

```
sax::/charlie\*.
```

Where you specify a file database name, the shell performs wildcard expansion. This means that it does not usually matter whether wildcards have preceding escape characters (back slashes). 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 **osls**; **osls** determines that they represent file pathnames, 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 pathname, and passes it to the shell command **ls**, which then does the expansion.

Executable Pathnames

Pathnames 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**, **osmv**, **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 (except **oscp** and **osmv**) pass the name on to the corresponding system command. Those commands that modify an individual database file in any way (**oschgrp**, **oschmod**, **oschown**, **osmv**, 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 what type of file it is.

Note: ObjectStore utilities return a zero 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 69.

All Server parameters have default values. Object Design 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.
- You can create a parameter file.

After you modify a Server parameter, you must shut down and restart the Server for any changes to parameters to take effect.

Using the osconfig Utility

When you create a rawfs with **osconfig rawfs**, the utility prompts you for a partition name. This results in a modification to the Server parameters.

When you run **osconfig**, the utility prompts you for the name of the Server log file. If you do not accept the default, this also results in a modification to the Server parameters.

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.
- 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 example below provides definitions for the **Log File**, **Partition***N*, and **Authentication Required** parameters:

```
Log File: /support2/local/ostore/sun4.r300/.log
Partition0: /support2/local/ostore/sun4.r300/.part0
Authentication Required: Unix Login
```

Search order

When you start an ObjectStore Server, ObjectStore looks for a parameter file in the following places in the order shown below. 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 name of the local host.

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. However you start the Server, you can configure it with command-line options and parameter file options that the Server reads when it starts.

The pathname of the Server executable is **$OS_ROOTDIR/lib/osserver.** See the installation instructions for setting **OS_ROOTDIR**).

Syntax

$OS_ROOTDIR/lib/osserver [*options*]

Options

Ordinarily, you use Server parameters to control Server behavior. You can also specify options when you execute the **osserver** utility.

-c Checkpoint. Forces all data to be propagated from the log to the database. The Server does not start after this checkpoint.

-d *int* Starts the Server in debug mode. Specify an integer from 1 through 50. The larger the number, the more information ObjectStore provides. You can also specify the **-F** option so that ObjectStore displays the information on the screen.

ObjectStore copies debug output to the **/tmp/ostore/oss_out** file, unless you redirect it to another file.

-F Foreground. Runs the Server process in the foreground. ObjectStore displays the information on the screen.This reverses the normal behavior, where the Server runs as a background process.

When run without **-F**, **osserver** returns **0** from a successful start-up of the background daemon; otherwise it returns **1**.

-i Initializes the Server log file and the rawfs, if you have one, with a confirmation prompt. Use with caution.

-I	(Uppercase I) Initializes the Server log file and the rawfs, if you have one, without a confirmation prompt. Use with extreme caution.
-p *pathname*	Specifies a file containing Server parameter settings that override the default settings. If you do not specify this option, ObjectStore uses only the default parameter file **$OS_ROOTDIR/etc/host_server_parameters**.
-v	Displays Server parameter values at start-up.

Description

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.

Debug mode

Output from debug mode can be very useful to Object Design personnel trying to help you solve a problem. 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 it down but does not affect clients. You should run the Server in debug mode only when you are experiencing a problem. In most cases, you want to specify **-d 50** so you can obtain all possible information.

Use -i and -I with caution

Be sure to move all data out of the log before you initialize it. See ossvrchkpt: Moving Data Out of the Server Transaction Log on page 253. Be sure to back up all data in the rawfs before you initialize it. See osbackup: Backing Up Databases on page 139.

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. When the log is in the rawfs, this frees space for use by other files. When the log is in the native file system, this reallocates space dedicated to the transaction log. The size of the log does not change.

Initializing when you have a rawfs

If you have a rawfs, specifying the **-i** or **-I** option with **osserver** initializes the rawfs and the transaction log. When you specify **-i**, ObjectStore responds with

```
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.

Initializing when there is no rawfs

If you do not have a rawfs, then specifying the **-i** or **-I** option with **osserver** initializes only the transaction log. When you specify **-i**, ObjectStore responds with

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.

Nonroot Server Start-Up

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 on page 87.

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 pathname for the partition and then initializes it. See your installation guide for details.

Restrictions

A file partition can be no larger than two gigabytes. There are no ObjectStore restrictions on the size of a rawfs partition; however, the allowable size depends on the operating system. Most 32-bit systems do not allow file partitions greater than 2 GB.

On SGI with XFS, file partitions can be greater than 2 GB.

The maximum number of partitions for a single Server is 32.

Specifying the Partitions in a Rawfs

The **Partition***N* Server parameter specifies the partitions in the rawfs. You add a partition to the rawfs by adding a **Partition***N* statement to the Server parameter file. Each **Partition***N* statement has the following form:

Partition*N***:** *type pathname* **{expandable | nonexpandable}**

N

Specifies a positive integer from 0 to *n*. **Partition***N* 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.

The value for type can be **PARTITION** or **UNIX** (except on AIX, HP–UX, and SGI IRIX; see below). If you specify **PARTITION**, 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. If you specify **UNIX**, the value of *pathname* is the absolute pathname of a UNIX file that is 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**, **LV**, or **RAW.**

- **UNIX** indicates the rawfs is stored in an AIX file.
- **LV** indicates the rawfs is stored in an AIX logical volume.
- **RAW** indicates the rawfs is stored in an AIX physical volume (raw disk).

HP–UX

On HP–UX, the value for *type* can be **UNIX**, **PARTITION**, or **DISK**.

- **UNIX** indicates the rawfs is stored in an HP–UX file.
- **PARTITION** indicates the rawfs is stored in an HP–UX logical volume.
- **DISK** indicates the rawfs is stored on a raw device.

SGI IRIX

On SGI IRIX, the value for *type* can be **UNIX**, **LV**, or **PARTITION**.

- **UNIX** indicates the rawfs is stored in an SGI IRIX file.
- **LV** indicates the rawfs is stored in an SGI IRIX logical volume.
- **PARTITION** indicates the rawfs is stored on a raw device.

pathname

Specifies the pathname 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 pathname.

AIX	On AIX, specify • An absolute pathname if you specify **UNIX** for type • The LV name if you specify **LV** for type, for example, **oslv** • The PV name if you specify **RAW** for type, for example, **hdisk6**
HP–UX	On HP–UX, specify • An absolute pathname if you specify **UNIX** for type. • The LV 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 pathname for the raw device if you specify **DISK** for type. For example, **/dev/dsk/c207d6s0**.
SGI IRIX	On SGI IRIX, specify • An absolute pathname if you specify **UNIX** for type. • The pathname of the raw device associated with the logical volume if you specify **LV** for type. For example, **/dev/rdsk/lv0**. • The pathname of the raw device if you specify **PARTITION** as type. For example, **/dev/rdsk/dks1d457**
expandable | **nonexpandable**	Indicates whether the partition is expandable in size or nonexpandable. By default, raw partitions are nonexpandable and file partitions are expandable.

It is unnecessary to specify more than one file in the same partition.

Modifying Partition Size

You can expand raw partitions only by setting aside additional raw space, adjusting the **Partition***N* statements as needed, and then restarting the Server.

Rawfs expansion

When you use files in a rawfs, you can control expansion of the rawfs as follows:

- If one file is used, it expands dynamically as needed, as long as there is room in the partition containing the file.
- If multiple files or a mixture of files and raw partitions is used for the rawfs, ObjectStore treats them differently with regard to expansion.
  - You can expand raw partitions only when you start the Server.
  - Files can expand dynamically, if you specify **expandable** in the **Partition***N* statement.

Example

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
```

Increase partition size

To increase the size of one or more existing partitions, check the sizes of the partitions with the **dkinfo**(8) 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) or the **osbackup** and **osrestore** utilities. Then substitute each new name for the corresponding old one in the Server parameter file.

Shrink rawfs

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.
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 328.
4. As **root**, restart the Server with the **osserver** utility.

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 32.

Setting Cache Manager Parameters

If a Cache Manager is not already running, the Cache Manager 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**
- **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.

Specifying the 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.

If the environment variable **OS_CACHE_DIR** is set, ObjectStore uses that directory. Cache and commseg files should be in the same directory.

Make sure that this directory exists. The Cache Manager does not automatically create it.

Specifying the 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.

If the environment variable **OS_COMMSEG_DIR** is set, ObjectStore uses that directory. Cache and commseg files should be in the same directory.

Specifying the 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 of **0** means that no limit is enforced.

Specifying the Mount Table Pathname Parameter

Default: **/etc/mtab**

The **Mount Table Pathname** parameter specifies the pathname of the mount table. The mount table indicates which UNIX file systems 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

This parameter instructs ObjectStore to use a mount table that is not in the default location. For HP–UX and System V platforms, except SGI, the default pathname is **/etc/mnttab**.

Object Design 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.

Specifying the 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 in order 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 start-up slower by forcing ObjectStore to create new cache and commseg files.

This parameter allows you to prevent ObjectStore cache files from using too much disk space.

The default of **0** means that no limit is enforced.

Specifying the Temporary Files Permission Parameter

When **f** 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 this 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 **$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.

2 **$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. Here 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
```

Increasing the Size of the Cache

A larger cache size does not necessarily improve performance.

On UNIX, an ObjectStore cache is a memory mapped file. Now, suppose you have a system where 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.

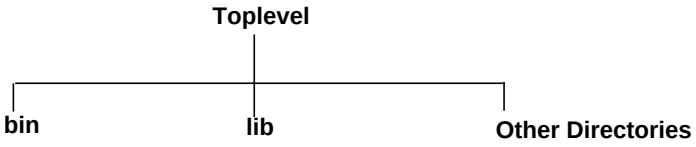
A program that would benefit from having a large cache would be one that accesses a large number of objects (pages) and then reaccesses the same objects before replacing the objects (pages) in the cache with other objects (pages) fetched from the database. Of course, as the cache size starts to become 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/segmented to achieve maximum performance is usually very application specific.

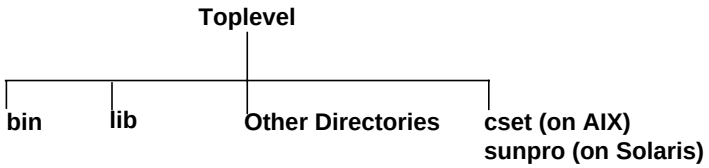
See **OS_CACHE_SIZE** on page 98 for information on setting the cache size.

Description of ObjectStore Directories

The ObjectStore directory structure appears in the following illustration. You must set **OS_ROOTDIR** to Toplevel.



The directory structure for AIX and Solaris 2 each contain one extra directory, **sunpro** on Solaris 2 and **cset** on AIX as shown in the following illustration. Each of these two directories contain links to the top level, **bin**, **lib** and **Other Directories.**



The **cset** and **sunpro** directories provide backward compatibility. In previous releases, the AIX and Solaris 2 platforms supported two compilers, a native C++ compiler and **Cfront**. Some components were common to the two compilers and some were compiler-dependent. The **cset** and **sunpro** directories contained compiler-specific components, and the common components were contained in a directory called **common**.

In the modified directory structure of ObjectStore Release 5.1 the common directories have been eliminated, and the **sunpro** and **cset** directories contain links to **bin**, **lib**, and so on. For backward compatibility, you can set **OS_ROOTDIR** to **Toplevel/cset** or **Toplevel/sunpro**.

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/ostore/osc4_out**
Server	**/tmp/ostore/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 under certain unusual error conditions. In these cases, this information can be helpful in understanding and resolving an error. When you report to Object Design a problem that might involve one of these daemons, 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.

Using Tapes with the osbackup Utility

When you execute the **osbackup** utility, you can use standard UNIX tape drives, including quarter-inch cartridge and 8mm cartridge drives. Some standard tape formats and sizes are

Format	***Capacity in Megabytes***
QIC–11	30
QIC–24	60
QIC–150	150
QIC–525	525
EXB–8200	2200
EXB–8500	5000

ObjectStore Use of /tmp/ostore

ObjectStore uses **/tmp/ostore** as the default location for the following files:

File	*For Information See*
Cache	**OS_CACHE_DIR** on page 98
Commseg	**OS_COMMSEG_DIR** on page 101
oss_out Server output file **osc4_out** Cache Manager output file	Finding Files Containing ObjectStore Messages on page 339
UNIX domain sockets	Modifying Network Port Settings on page 51

AIX Considerations

The following information provides additional considerations for ObjectStore Release 5 on AIX.

Using SCSI Tape Drives

When using tapes with the **osbackup** and **osrestore** utilities, you have several choices of how to configure 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. The **osbackup** utility cannot work when the block size is greater than 512 bytes.

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 Release 5

In the event that you want to completely remove ObjectStore Release 5.0 from your AIX system, 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, 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 NT and Windows 95. For complete information, you should consult the first six chapters of this book along with this chapter.

The topics covered include

Using ObjectStore Utilities

During installation, you use the ObjectStore Setup Utility (**SETUP.EXE**) to configure ObjectStore daemons and applications — the Server, Cache Manager, Database Manager, and Browser.

All utility executables are stored in ObjectStore's **BIN** directory.

Windows NT

On Windows NT, the **INSTALL** program installs the Server and Cache Manager daemons as NT 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 using the Service Control Dialog, located in the Control Panel **Services** icon.

On Windows NT, run ObjectStore utilities in a command prompt window.

Windows 95

On Windows 95, the Server daemon is added to the system's Start-up program group. This daemon starts automatically when you start up Windows. The Cache Manager is launched automatically from the first client program you run. You can also start the ObjectStore daemons in an MS–DOS prompt window, or by double-clicking on the icons in the ObjectStore folder, or by using the **Start | Run** command.

Exceptions and Memory Leaks

Some Windows NT functions allocate memory in such a way that it is not freed if an exception is signaled inside a callback function, but is handled outside the Windows NT function. This can cause memory leaks.

Memory Requirements for Windows 95

Object Design recommends a minimum of 16 MB memory on Windows 95 systems. The ObjectStore Server is tuned for high performance on network servers. Because of this, it sometimes does not run efficiently on systems with less than 16 MB physical memory. If you use less than 16 MB physical memory, read this section carefully to ensure that your system can run ObjectStore properly. The information here applies to Server-only and client-only installations. The goal is to minimize memory consumption.

Performance

With less than 16 MB physical memory, ObjectStore's performance might be degraded, and in some cases Windows 95 itself might be unreliable.

Unsupported configuration

You should never run a network server on Windows 95 with less than 16 MB physical memory; Object Design does not support this configuration. Stand-alone systems with less than 16 MB physical memory require careful tuning to achieve adequate performance and efficient resource consumption.

Run only ObjectStore clients

Generally, systems that run only ObjectStore clients have the fewest problems with memory consumption.

Virtual memory requirements

The sum of your physical memory and the swap file size must be at least 24 MB. For example, if you have 8 MB physical memory, your swap file must be at least 16 MB. In all cases, Object Design recommends a minimum 8 MB swap file. Set up the Windows 95 swap file in the Control Panel **System** icon.

Modifying Server parameters

When running an ObjectStore Server on a stand-alone system with less than 16 MB physical memory, you can reduce the Server's memory consumption by setting the following Server parameters as indicated in the following table. You must restart the Server for the new parameter settings to take effect.

Parameter	***Setting***
Log Record Segment Buffer Size	256 sectors (default is 1024)
Max Data Propagation Per Propagate	256 sectors (default is 256)
Propagation Buffer Size	512 sectors (default is 8192)

Modifying client environment variables

You can reduce memory consumption by setting the **OS_CACHE_SIZE** environment variable to an amount less than its default size of 8 MB (8388608). The minimum size on Windows 95 is 480 KB (491520). Generally, on systems with 12 MB physical memory, 3 MB (3145728) is a good value. Some applications that access large amounts of data simultaneously might require larger values.

The client cache is only filled as persistent data is actually allocated or referenced. If your application references only a small amount of persistent data, reducing **OS_CACHE_SIZE** will probably have little or no effect.

Reducing **OS_AS_SIZE** and / or **OS_COMMSEG_SIZE** to conserve memory is generally not worthwhile.

Specifying File Database Pathnames

You specify a file database with an operating system pathname. For example:

```
os_database::open("d:\\carter\\myfiledb")
```

UNC pathname

The syntax for specifying a UNC pathname is

```
os_database::open("\\\\servername\\sharename\\myfiledb")
```

In C and C++, you must escape the back slash (\) with another back slash.

You can also specify a relative pathname.

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 281.

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 pathname is called a *Server-relative* pathname. 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.

Setting Server Parameters

The Server, Cache Manager, and Browser store their parameters in a central location.

Each Server parameter that you can specify is described in Chapter 2, Server Parameters, on page 69.

Windows

On Windows NT and Windows 95, parameters are in the Windows registry database.

Changing parameters

You can edit these parameters using the ObjectStore utility **SETUP.EXE**. **SETUP.EXE** allows you to modify all fields except for license information. ObjectStore license information is written by the **INSTALL** program.

To set Server parameters, run **SETUP.EXE**, select **Advanced Options**, and then select **Server Parameters**. See your ObjectStore installation guide for 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 start-up.

Windows NT

On Windows NT, you can configure the Server as an NT service. This is the preferred method.

When starting the Server as an NT service, you can specify **osserver** command-line options in the following table in the NT service administration dialog.

On Windows NT, you can run the Server as an NT console application. In this case, you run **OSSERVER.EXE** from an NT command window, passing **-console** as the first argument.

At start-up, ObjectStore configures the Server according to parameter options in the registry database.

Windows 95

On Windows 95, you can start the Server manually by double-clicking on the ObjectStore Server icon in the ObjectStore folder, or by using **Start | Run,** or from an MS-DOS prompt window.

Command-line options

Ordinarily, you use Server parameters to control the Server's behavior. However, you can also specify the following command-line options to **osserver**.

-c	Checkpoint. Forces all data to be propagated from the log to the database. The Server does not start after this checkpoint.
-con or **-console**	Windows NT only: runs the Server as a console-mode application. If you specify this option, it must be the first option you specify.

-d *int* — Starts the Server in debug mode. Specify an integer from 1 through 50. The larger the number, the more information ObjectStore provides. If the ObjectStore Server Service is running, you must stop it and restart it. Use the Services applet in the Control Panel.

Another alternative is to stop the ObjectStore Server Service. Then you can specify the **-con** option so that ObjectStore displays the information on the screen.

Debug output goes to **%OS_TMPDIR%\osserver.txt**.

-i — Initializes the Server log file and the rawfs, if you have one, with a confirmation prompt. Use with caution.

-I — (Uppercase I) Initializes the Server log file and the rawfs, if you have one, without a confirmation prompt. Use with extreme caution.

-v — Displays Server parameter values at start-up.

Use -i and -I with caution

Be sure to move all data out of the log before you initialize it. See ossvrchkpt: Moving Data Out of the Server Transaction Log on page 253. Be sure to back up all data in the rawfs before you initialize it. See osbackup: Backing Up Databases on page 139.

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. When the log is in the rawfs, this frees space for use by other files. When the log is in the native file system, this reallocates space dedicated to the transaction log. The size of the log does not change.

Initializing when you have a rawfs

If you have a rawfs, then specifying the **-i** or **-I** option with **osserver** initializes the rawfs and the transaction log. When you specify **-i,** ObjectStore responds with

> 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.

Initializing when you do not have a rawfs

If you do not have a rawfs, then specifying the **-i** or **-I** option with **osserver** initializes only the transaction log. When you specify **-i,** ObjectStore responds with

```
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.

Start-up message

After starting, the Server displays the message Server started to let you know that it is ready to accept requests from clients on the network.

Troubleshooting Windows NT

You might have a situation where you cannot run the Server and the Cache Manager as NT 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 if the Server and Cache Manager can be started in console mode using the **-con** option to **osserver**. If they start properly in console mode, this 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 NT Services.

1. If you ran an earlier ObjectStore release, confirm that it was properly uninstalled prior to installing the new release. This is necessary so that the image path is properly set for the NT Services.
2. Ensure that ObjectStore was installed using an account with Administrative privileges.
3. To run the Server and Cache Manager as NT 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 R4.0

Then, check that the setting for the image path is correct. Do the same for **ObjectStore Server R4.0**.

6 Check 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 using the **-con** option), then it is not a permission problem.

If it is not a permission problem, then 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 start-up output in debug mode. If this is not the case, then 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 Object Design 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 30.

Creating a rawfs

To create the rawfs or add partitions to an existing rawfs, use the **SETUP.EXE** utility. When you create the rawfs, you specify a disk drive letter and a file name. After you create the rawfs, you can add file partitions. See your installation guide for details.

Using SETUP.EXE to Add File Partitions

Be sure to shut down the Server before you run **SETUP.EXE**. You cannot run **SETUP.EXE** while the Server is running.

The **Partition***N* Server parameter specifies the partitions in the rawfs. It creates the Server partition file if it does not exist, using the pathname given in the **Partition***N* statement. You add a partition to the rawfs by specifying a **Partition***N* statement when prompted to by the **SETUP.EXE** utility. Each **Partition***N* statement has the following form:

Syntax

Partition*N* **FILE** *pathname* **{expandable | nonexpandable}**

N	Specifies a positive integer from 0 to *n*. **Partition***N* statements can appear in the Server parameters 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.
FILE	Specifies that the partition is a file partition. Required.
pathname	Specifies the absolute pathname of the partition. It must begin with a drive letter, colon, and back slash.
expandable | **nonexpandable**	Indicates whether or not the size of the partition can increase. Required.

It is unnecessary to specify more than one file in the same drive letter.

Example

```
Partition0 FILE c:\mypart0 expandable
Partition1 FILE d:\mypart1 expandable
Partition2 FILE e:\bigpart99 expandable
```

Modifying Partition Size

Controlling expansion

You can control expansion of the rawfs as follows:

- If one file is used, it expands dynamically as needed, as long as there is room in the partition containing the file.
- If multiple files are used for the rawfs, they can expand dynamically, if you specify **expandable** in the **Partition***N* statement.

For example, suppose the rawfs contains three files (**\usr1\one**, **\usr2\two**, and **\usr3\three**). If you want to permit only two of the files to expand, use **SETUP.EXE** to specify the partitions as follows:

Example

```
Partition0: FILE c:\usr1\one expandable
Partition1: FILE d:\usr2\two expandable
Partition2: FILE e:\usr3\three nonexpandable
```

When you exit from **SETUP.EXE**, it prompts you to indicate whether or not you want to restart the Server. If you indicate **No**, you must run **osserver** to start the Server.

Starting the Cache Manager

Windows NT

The Cache Manager is an NT 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 Control Panel Services applet to configure the Cache Manager for Manual Start-up.

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 Start-up.

If you want to start the Cache Manager as a console application, use the following command:

oscmgr4 -con 0 [*debug_level*]

Windows 95

On Windows 95, the Cache Manager is started automatically as needed. If you need to start it manually, use **Start | Run | oscmgr4 0** [*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	**OSCMGR4.TXT**

If the file does not already exist, ObjectStore creates it; if the file does exist, ObjectStore appends 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()**.

ObjectStore daemons seldom send messages to these files except under certain unusual error conditions. In these cases, this information can be helpful in understanding and resolving an error. When you report to Object Design a problem that might involve one of these daemons, 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 NT prompts for a name and password. See Authentication Required on page 73.

If you want the ObjectStore client to use RPC authentication, use the Windows NT **REGEDT32** utility, and set the following variables, where *x.x* is the ObjectStore release number, and *username* is your username:

HKEY_LOCAL_MACHINE\Software\Object DesignInc.\ObjectStore*x.x*\Remote*username*\UNIX.UID
HKEY_LOCAL_MACHINE\Software\Object DesignInc.\ObjectStore*x.x*\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 NT account with Administrator privileges.

Note that the **HKEY_LOCAL_MACHINE\Software\Object Design Inc.\ObjectStore.4.0\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 still must enter a user name / password to access the Server.

To prevent this, you can change the **HKEY_LOCAL_MACHINE\Software\Object Design Inc.\ObjectStore.4.0\Remote** to have read-only privileges for users with non-Administrator privileges. You can set the privileges using the **Security | Permissions** dialog in the registry editor.

About Client/Server Communication

Windows clients and Servers can use three network layers:

- Within a single machine, ObjectStore uses an interprocess communication mechanism based on named shared memory objects.
- Windows Sockets provide TCP/IP connections. Object Design tests and supports a limited number of Windows Sockets implementations.

A Server can and typically does serve all three networks simultaneously. A client chooses the first network available that recognizes the name of the Server host.

For more information, see *ObjectStore Installation and License for Windows*, Network Support.

Using an NT Server to Access Remote Databases

Netware

ObjectStore can be used with the Gateway Services for NetWare GSNW feature.

All Remote Hosts

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 NT as well as NetWare file hosts.

Access Control for Remote Databases

Windows NT refuses to allow the Server to use credentials obtained from a remote client host on a remote file host. There are two alternatives to deal with this issue. They are described in Microsoft Knowledge Base article Q132679, available on the Microsoft TechNet CD, or on the web as

http://www.microsoft.com/kb/bussys/winnt/q132679.htm

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.

Chapter 9
Managing ObjectStore on OS/2

This chapter provides information for managing ObjectStore on OS/2 systems. For complete information, you should consult the first six chapters of this book along with this chapter.

The topics discussed are

Specifying File Database Pathnames

You specify a file database with an operating system pathname. For example:

```
os_database::open("d:\\carter\\myfiledb")
```

UNC pathname

The syntax for specifying a UNC pathname is

```
os_database::open("\\\\servername\\sharename\\myfiledb")
```

In C and C++, you must escape the back slash (\) with another back slash.

You can specify a relative pathname.

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 281.

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 an OS/2 host:	**elvis:D:\usr\barbar\employees**

This specifies a database, **employees**, in the directory **\usr\barbar** on the host **elvis**.

Setting Server Parameters

Each Server parameter that you can specify is described in Chapter 2, Server Parameters, on page 69.

OSTORE.INI

The ObjectStore Server stores its parameters in the **OSTORE.INI** file. This file is in the standard format for OS/2 parameters files. The ObjectStore installation and setup utilities create the **OSTORE.INI** file and store it, by default, in the **\OS2** directory on the boot drive.

OSSETUP

All Server parameters have default values. Object Design recommends that you use the default value for each parameter, which requires no action on your part. However, you can use the **OSSETUP** utility to modify parameters in the **OSTORE.INI** file. See your installation guide for information.

Using **-p**

If you do not want to use the Server parameters in the **OSTORE.INI** file, you can create a standard OS/2 parameters file that contains the parameter values you want. When you start the Server with the **osserver** utility, you can specify this Server parameters file with the **-p** option.

OS2.INI
OS2SYS.INI

Alternatively, you can remove the **OSTORE.INI** file and place Server parameters in the OS/2 user profiles, **OS2.INI** and **OS2SYS.INI**.

Search order

ObjectStore uses the following search order to find parameter settings:

1 File specified with the **-p** option to **osserver**

2 **OSTORE.INI** file created by **OSSETUP**

3 OS/2 user profiles (**OS2.INI** and **OS2SYS.INI**)

If ObjectStore does not find a parameters file, it displays a message.

After you modify a Server parameter, you must shut down and restart the Server for the parameter to take effect.

Starting the Server

You can start the Server

- Automatically when OS/2 is started. You specify this with **OSSETUP**, the ObjectStore setup utility. This is the usual method.
- From an icon.
- By running the **OSSERVER** utility.

Regardless of when it is started, you can configure the Server with various command-line options and parameter file options that are read at start-up.

Command-line options

When you start the ObjectStore Server with the **osserver** utility, you can specify the following options:

-c Forces all data to be propagated from the log to the database. The Server is not started following this checkpoint.

-d *int* Starts the Server in debug mode. Specify an integer from 1 through 50. The larger the number, the more information ObjectStore provides. You can also specify the **-F** option so that ObjectStore displays the information on the screen.

ObjectStore copies debug output to the **OSSERVER.TXT** file, unless you redirect it to another file.

-F Foreground. Runs the Server process in the foreground. This reverses the normal behavior, where the Server runs as a background process.

-i Initializes the Server log file and the rawfs, if you have one, with a confirmation prompt. Use with caution.

-I (Uppercase I) Initializes the Server log file and the rawfs, if you have one, without a confirmation prompt. Use with extreme caution.

-p *pathname* Specifies a file containing parameter settings that override those in **OSTORE.INI**.

-v Shows Server parameter values at start-up.

After starting, the Server displays the message Server started to let you know that it is ready to accept requests from clients on the network.

Use -i and -I with caution

Be sure to move all data out of the log before you initialize it. See ossvrchkpt: Moving Data Out of the Server Transaction Log on page 253. Be sure to back up all data in the rawfs before you initialize it. See osbackup: Backing Up Databases on page 139.

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. When the log is in the rawfs, this frees space for use by other files. When the log is in the native file system, this reallocates space dedicated to the transaction log. The size of the log does not change.

Initializing when you have a rawfs

If you have a rawfs, then specifying the **-i** or **-I** option with **osserver** initializes the rawfs and the transaction log. When you specify **-i,** ObjectStore responds with

> 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.

Initializing when you do not have a rawfs

If you do not have a rawfs, then specifying the **-i** option with **osserver** initializes only the transaction log. When you specify **-i,** ObjectStore responds with

> 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.

Starting Cache Manager

The Cache Manager runs automatically when it is needed; no command is necessary to start it. However, if you do want to start the Cache Manager explicitly, use this command:

```
START /N OSCMGR4 0
```

Controlling access to Server

The host list on an OS/2 Server determines which hosts can access databases.

Using OS/2 Environment Variables

You can use these NFS client software variables

UNIX.GID

Default: not set

For ObjectStore on OS/2, specifies the group ID (GID) for a UNIX client process.

UNIX.UID

Default: not set

Specifies the UNIX user ID (UID) for a client process being accessed from an OS/2 Server.

Specifying Utility Names

FAT names

The File Allocation Table (FAT) file system restricts file names to an eight-character file name and three-character file name extension. Consequently, ObjectStore utility names are truncated to eight characters when ObjectStore is installed on a FAT file system.

HPFS names

When installed on a High Performance File System (HPFS), ObjectStore utilities use their full command names.

This book refers to ObjectStore utilities by their full names.

Utility executables

All utility executables are stored in ObjectStore's **BIN** directory. For example:

%OS_ROOTDIR%\BIN\OSSERVER.EXE

Finding Files Containing ObjectStore Messages

In some cases, when an ObjectStore daemon process reports an error, ObjectStore sends the output to a file. These files are

Server errors	**OSSERVER.TXT**
Cache Manager errors	**OSCMGR4.TXT**

If the file does not already exist, ObjectStore creates it. If the file does exist, ObjectStore appends to it.

ObjectStore uses the path assigned to one of the environment variables listed below to determine the directory in which to place these files. It searches in the following order:

Search order

1. **%OS_TMPDIR%**
2. **%TEMP%**
3. **%TMP%**

ObjectStore daemons seldom send messages to these files except under certain unusual error conditions. In these cases, this information can be helpful in understanding and resolving an error. When you report to Object Design a problem that might involve one of these daemons, find such a file, if it exists, and provide the contents.

Deleting error files

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.

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 30.

Creating a rawfs

To create the rawfs or add partitions to an existing rawfs, use the **OSSETUP** utility. When you create the rawfs, you specify a disk drive letter and a file name. After you create the rawfs, you add Server partitions. See your installation guide for details.

Using OSSETUP to Add File Partitions

Be sure to shut down the Server before you run **OSSETUP**. You cannot run **OSSETUP** while the Server is running. When you exit from **OSSETUP**, it prompts you to indicate whether or not you want to restart the Server. If you indicate **No**, you must run **osserver** to start the Server.

The **Partition***N* Server parameter specifies the partitions in the rawfs. It creates the Server partition file if it does not exist, using the pathname given in the **Partition***N* statement. You add a partition to the rawfs by specifying a **Partition***N* statement when prompted to by the **OSSETUP** utility. Each **Partition***N* statement has the following form:

Syntax

Partition*N* **FILE** *pathname* **{expandable | nonexpandable}**

N	Specifies a positive integer from 0 to *N*. **Partition***N* 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.
FILE	Specifies that the partition is a file partition. Required.
pathname	Specifies the absolute pathname of the partition. It must begin with a back slash.
expandable | **nonexpandable**	Indicates whether or not the size of the partition can increase. Required.

It is unnecessary to specify more than one file in the same partition.

Example

```
Partition0 FILE c:\mypart0 expandable
Partition1 FILE d:\mypart1 expandable
Partition2 FILE e:\bigpart99 expandable
```

Modifying Partition Size

Controlling expansion

You can control expansion of the rawfs as follows:

- If one file is used, it expands dynamically as needed, as long as there is room in the partition containing the file.
- If multiple files are used for the rawfs, they can expand dynamically, if you specify **expandable** in the **Partition***N* statement.

For example, suppose the rawfs contains three files (**\usr1\one**, **\usr2\two**, and **\usr3\three**). If you want to permit only two of the files to expand, use **OSSETUP** to specify the partitions as follows:

Example

```
Partition0: FILE c:\usr1\one expandable
Partition1: FILE d:\usr2\two expandable
Partition2: FILE e:\usr3\three nonexpandable
```

Capturing Debug Information

If you need help from Object Design support, follow these steps to capture debug information.

1 On all machines, create the **c:\temp** directory if it is not already there.

2 Set the **OS_TMPDIR** environment variable to **c:\temp** by adding the following line to the **config.sys** file:

SET OS_TMPDIR=c:\temp

3 Reboot all machines.

ObjectStore looks for **OS_TMPDIR**, then **TEMP**, and finally **TMP**. If none of these are defined, ObjectStore writes to the boot drive.

4 On the Server machine, open a window and stop the Server:

ossvrsht -f *hostname*

5 Set the **OS_DEBUG_NETWORK** environment variable to **1**.

6 Start the Server with the following command:

osserver -F -v -d 10

You do not need to redirect the output. ObjectStore places the output in the **oss.out** and **osserver.txt** files in the **c:\temp** directory, using the **OS_TMPDIR** environment variable.

The **osserver** options have the following meanings:

-F	Runs the Server in the foreground.
-v	Runs the Server in verbose mode. This outputs a lot of information about what the Server is doing.
-d	Indicates that the following number is a debug level.
10	Specifies the debug level. The debug levels that you typically use are **1**, **3**, **5**, or **10**, depending on the level of detail required for the trace.

7 On each client,

a Open a window and shut down the Cache Manager with the **oscmshtd** utility.

b Set the **OS_DEBUG_NETWORK** environment variable to **1**.

c Start the Cache Manager with the command

oscmgr4 0 50 0 1 1

You do not need to redirect the output. ObjectStore uses **OS_TMPDIR** to send the output to the **osc4.out** file. The **oscmgr4** options have the following meanings:

0	Indicates that the Cache Manager was not automatically loaded with a previous connection to the first client.
50	Specifies the Cache Manager debug level.
0	Required queue length.
1	Turns on debug messages for individual Cache Manager threads.
1	Sends debug output to the **osc4.out** file.

Before activating any client, open a window and set the environment **OS_DEBUG_NETWORK** variable to **1** in that window.

8 From that window, run the client to produce the error, redirecting the network debug information to a file in order to capture it.

9 Send the following output to Object Design:

- For the Server, located in the **c:\temp directory**, the files **oss.out**, **osc4.out**, and **osserver.txt**
- For each client, located in its **c:\temp** directory, the file **osc4.out**
- For each client, the redirected network debug output

File Locking with NFS

You can access a remote database with the syntax

host:*drive*:*path*

However, the IBM NFS implementation for OS/2 does not allow locking a file across an NFS volume. Doing so is not safe for databases that are remote to the ObjectStore Server. But it is quite safe if the database is physically located on the ObjectStore Server.

For example, a client can access a database by means of NFS. The client NFS mounts the drive from the ObjectStore Server so that it appears as a local drive on the client. Now the client can reference the database as, for example,

g:\my\path\to\database

ObjectStore translates that path to something the ObjectStore Server can understand, and since it is physically on the ObjectStore Server, it can locate the Server-local file.

There is another scenario that is not safe. In this case, the database is not physically resident on the ObjectStore Server. By default, ObjectStore does not allow this. You can, however, set up a locator file to allow it. See Chapter 5, Using Locator Files to Set Up Server-Remote Databases, on page 281. In this unsafe scenario, the database is physically resident on a third machine, the NFS server, which might be the client machine. The ObjectStore Server needs to lock the database by means of NFS. On OS/2, this is not possible. You cannot lock across NFS volumes, which makes it difficult to guarantee the integrity of the database.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W